



USE OF TRUST VECTORS IN SUPPORT OF THE CYBERCRAFT INITIATIVE

THESIS

Michael Stevens, Captain, USAF

AFIT/GIA/ENG/07-03

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this paper are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government, except where noted

USE OF TRUST VECTORS IN SUPPORT OF THE CYBERCRAFT INITIATIVE

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science

Michael Stevens, B.S.C.S.

Captain, USAF

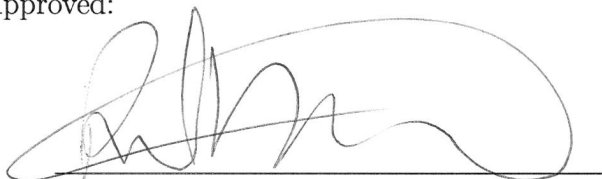
March 2007

USE OF TRUST VECTORS IN SUPPORT OF THE CYBERCRAFT
INITIATIVE

Michael Stevens, B.S.C.S.

Captain, USAF

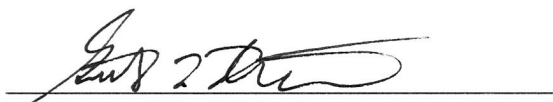
Approved:



Maj Paul D. Williams, PhD (Chairman)

6 MAR 07

date



Dr. Gilbert L. Peterson (Member)

6 MAR 07

date



Lt Col Stuart H. Kurkowski, PhD
(Member)

6 Mar 2007

date

Abstract

As the United States Air Force extends its mission into the realm of Cyberspace, the CyberCraft Initiative is designing a framework for command and control of future Air Force Cyber-weapon systems. These Cyber-weapon systems, CyberCraft, will autonomously operate and defend the Air Force networks and information systems to provide *Cyberspace Superiority* in support of the defense of the United States. The fundamental research question of the CyberCraft Initiative is “*What is required for a commander to trust a CyberCraft to autonomously defend military information systems?*”

The Trust Vector model [17] is one method of integrating trust into the CyberCraft fleet. Trust Vectors define trust and distrust between agents based on three components; current and historical data, intrinsic knowledge of the remote agent’s abilities, and recommendations from other agents. This research explores the suitability of the Trust Vector model to the CyberCraft Initiative and defines the strengths and remaining challenges of using the Trust Vector model.

This research finds that the Trust Vector model can be modified to integrate trust into the CyberCraft Initiative. Several expansions to the model are proposed, including applying the Trust Vector model to an asynchronous paradigm for data

transactions. This research also determines the limits of the utility of historical data for the Trust Vector model.

Acknowledgements

First and foremost, I owe a large debt of gratitude to

- AFRL\IFGB and the CyberCraft Initiative, for sponsoring this research
- Major Williams, for his vast support in the production of this Thesis
- and all the professors and instructors at AFIT for letting me ask “just one more question”
- and especially my wife and daughter, for supporting me on the home front

Michael Stevens

Table of Contents

	Page
Abstract	iv
Acknowledgements	vi
List of Figures	x
List of Tables	xii
List of Abbreviations	xiii
 I. Introduction	 1
1.1 The need for CyberCraft	4
1.2 Thesis Statement	6
 II. Background	 9
2.1 The CyberCraft Initiative	9
2.1.1 Components	10
2.1.2 Fundamental Problem Areas	13
2.2 Trust Vector Model	20
2.2.1 Trust Vectors Overview	23
2.2.2 Components	24
2.2.3 Trust Policy Vector	31
2.2.4 Degradation Function	32
2.2.5 Vulnerability Assessment Scenario	34
2.3 Other work in trust between computers	36
 III. Trust Vector Architecture	 46
3.1 Properties of the Trust Vector model	46
3.1.1 Knowledge Component	46
3.1.2 Transaction Paradigm	47
3.1.3 Transitivity of Trust	55
3.2 Recommended Settings and Modifications	55
3.2.1 Experience Component: Event Values	55
3.2.2 Experience Component: Intervals	56
3.2.3 Integration of Entities	58
3.2.4 Associated Risk of a Trust Relationship	58
3.2.5 Degradation Function	59

	Page
IV. Experimental Methodology	60
4.1 Experiment I: Limits of Historical Data for Calculating the Experience Component	60
4.1.1 Problem Statement	61
4.1.2 Hypothesis	61
4.1.3 Motivation	61
4.2 Experiment II: Utility of Degradation Function	61
4.2.1 Problem Statement	62
4.2.2 Hypothesis	62
4.2.3 Motivation	62
4.3 OS Fingerprinting Scenario	62
4.3.1 Experiment I: Scenario	63
4.3.2 Experiment II: Scenario	64
4.4 System Boundaries	64
4.4.1 Communications component	65
4.4.2 Data Input component	66
4.4.3 Decision component	66
4.4.4 Trust Vectors	67
4.4.5 Experiment I: System Boundaries	68
4.4.6 Experiment II: System Boundaries	69
4.5 System Services	69
4.6 Workload	70
4.6.1 Experiment I: Workload	70
4.6.2 Experiment II: Workload	70
4.7 Metrics	70
4.8 Parameters	71
4.9 Factors	73
4.10 Experiment I: Factors	74
4.11 Experiment II: Factors	74
4.12 Evaluation Technique	74
4.13 Experimental Design	74
V. Analysis and Results	75
5.1 Experiment I: Limits of Historical Data for Calculating the Experience Component	75
5.1.1 Analysis	75
5.1.2 Experiment I: Results	77
5.1.3 Results	80
5.2 Experiment II: Utility of Degradation Function	80
5.2.1 Analysis	80
5.2.2 Results	83

	Page
VI. Conclusions and Future Work	85
6.1 Experiments	86
6.1.1 Experiment I: Limits of Historical Data for Calculating the Experience Component	86
6.1.2 Experiment II: Utility of Degradation Function	87
6.2 Future Work	87
6.2.1 Transactional Paradigm	87
6.2.2 Number of Agents in a Recommendation Pool	88
6.2.3 Abstraction of Trust Context	88
6.2.4 Experience Component	89
6.2.5 Knowledge Component	89
6.2.6 Normalization of Recommendations	90
6.2.7 Additional Components	91
6.2.8 Differing Views of Data	92
6.2.9 Ambiguity of Data	92
6.2.10 Integration of VTrust into the CyberCraft Initiative	92
6.2.11 Lack of Corroborating Data and Markovian Decision Processes	93
6.2.12 Dynamic Routing and Topology Control	93
6.2.13 Evolutionary Algorithms and Artificial Immune Systems	94
6.2.14 The “High Ground” of Cyberspace	94
Bibliography	96

List of Figures

Figure		Page
1.1.	Artist's rendition of CyberCraft in Cyberspace.	3
2.1.	Different Mapping Contexts.	15
2.2.	Calculation of the experience component.	26
2.3.	Demonstration of the recommendation component.	29
2.4.	Trust degradation as time increases.	33
2.5.	Scenario of Agents 1, 2, and 3 scanning the same computer. . .	35
2.6.	Trust Based on Knowledge (Jøsang).	39
2.7.	Purser's Graphical Model of Trust.	40
3.1.	Transactional Paradigm: <i>Alice</i> posts data to the CyberCraft In- formation Store.	50
3.2.	Transactional Paradigm: <i>Bob</i> retrieves <i>Alice</i> 's data from the In- formation Store.	50
3.3.	Transactional Paradigm: <i>Cathy</i> also retrieves <i>Alice</i> 's Data from the Information Store.	51
3.4.	Transactional Paradigm: <i>Cathy</i> queries other agents for recom- mendations about <i>Alice</i>	51
3.5.	Transactional Paradigm: <i>Bob</i> , <i>David</i> , and <i>Ethel</i> send recom- mendations to <i>Cathy</i>	52
3.6.	Modification of the experience component: Event values.	56
3.7.	Modification of the experience component: Fixed time intervals.	57
4.1.	System Under Test: Implementation of the Trust Vector model.	65
4.2.	Experiment I Design.	68
4.3.	Experiment II Design.	69
5.1.	Calculation of experience component.	77
5.2.	Trust Value for different experience intervals kept.	78
5.3.	Higher values degrade faster using $\mathbf{v}(T_{t_n}) = \mathbf{v}(T_{t_i})e^{-(v(T_{t_i})\Delta t)^{2k}}$. .	81

Figure		Page
5.4.	All values degrade at an equal rate using $\mathbf{v}(T_{t_n}) = \mathbf{v}(T_{t_i})e^{-(v(T_{t_i})^{-1}\Delta t)^{2k}}$.	81
5.5.	Degradation of Trust Value of 1 at different rates.	82
5.6.	Degradation of trust at different levels of the <i>period of decay</i> (τ).	83

List of Tables

Table		Page
2.1.	Symbology used to describe the components of the Trust Vector Model.	25
2.2.	Symbology used to describe the Trust Policy Vector.	30
2.3.	Symbology used to describe the degradation function.	32
4.1.	Parameters of the Trust Vector Model.	72
5.1.	Decreasing value of Oldest Interval.	76
5.2.	Decreasing value of Oldest Interval.	80
6.1.	Modifications of the normalization equation.	90
6.2.	Effect of different normalization equations.	91

List of Abbreviations

Abbreviation		Page
USAF	United States Air Force	1
IDS	intrusion detection systems	1
ECM	Electronic Counter-Measure	4
INFOCON	Information Operations Condition	4
SQL	Standard Query Language	4
AFRL	Air Force Research Laboratory	6
FAA	Federal Aviation Administration	6
OS	Operating System	10
C2	Command and Control	10
COP	Common Operating Picture	13
AV	Anti-Virus	13
NIPRNET	Non-Secure Internet Protocol Router Network	14
SIPRNET	Secure Internet Protocol Routing Network	14
C3	Command, Control, and Communications	16
TPM	Trusted Platform Module	19
P2P	peer-to-peer	41
ATC	Air Traffic Control	48
DHT	Distributed Hash Table	48
CD	compact disc	49
AFIT	Air Force Institute of Technology	94
AIS	artificial immune systems	94

USE OF TRUST VECTORS IN SUPPORT OF THE CYBERCRAFT INITIATIVE

I. Introduction

On 7 December 2005, Secretary of the Air Force Michael Wynne and Chief of Staff of the Air Force T. Michael Moseley stated “The mission of the United States Air Force (USAF) is to deliver sovereign options for the defense of the United States of America and its global interests – to fly and fight in Air, Space, and Cyberspace” [22]. Various Air Force units have been protecting Air Force networks through use of network policies and technological means (e.g., firewalls and intrusion detection systems (IDS)) for a number of years, but this formal expansion of the Air Force Mission into the domain of Cyberspace comes with additional challenges. These challenges spring from the properties of Cyberspace, which include but are not limited to [10]:

Non-jurisdictional: There is no governing authority over Cyberspace. If an Air Force base was targeted by an air attack, we could track the offending aircraft back to its country of origin, and we could then engage with that country militarily or diplomatically. With a Cyber-attack, not only does the relative anonymity of the internet make it extremely difficult to attribute the attack to a geographical location, but laws regarding the use of computers vary between

countries, so what one country may interpret as an act of war the offending country might interpret the act as only a misdemeanor offense.

Low cost of entry: The price tag for a competent air force or blue-water navy in terms of training, personnel, and equipment is steep enough to discourage most nations from creating a viable threat to the United States military. Conversely, to build a threatening Cyber-warfare squadron requires a few personnel with computer science backgrounds and a handful of inexpensive computers. These Cyber-warfare squadrons could cripple an adversary that is militarily more powerful.

Non-state players: Complementing the previous property, organized crime and terrorist groups can build their own Cyber-warfare unit and attack nation-states just as a third world nation's Cyber-warfare squadron could attack the U.S., European Union, or People's Republic of China.

Interdependance: One adage about network security is "*A vulnerability assumed by one is shared by all*". An attacker does not need to attack all aspects of a target network, the attacker only needs to find one vulnerability to exploit (much as a burglar only needs to find one unlocked window to gain entry). The defender needs to protect the entire attack surface and worry about insider threats. Air Force base networks share information between themselves and trust the security of other base networks. If an attacker can compromise the security of one base network, the attacker can then access the other base networks of the Air Force with the trust given to the compromised base.

To meet these challenges, the CyberCraft Initiative is creating a set of standards for future cyberspace weapon systems¹, which will be used to provide *Cyberspace Superiority* to U.S. forces, much as aircraft provide *Air Superiority* and spacecraft (satellites) provide *Space Superiority* to U.S. forces.

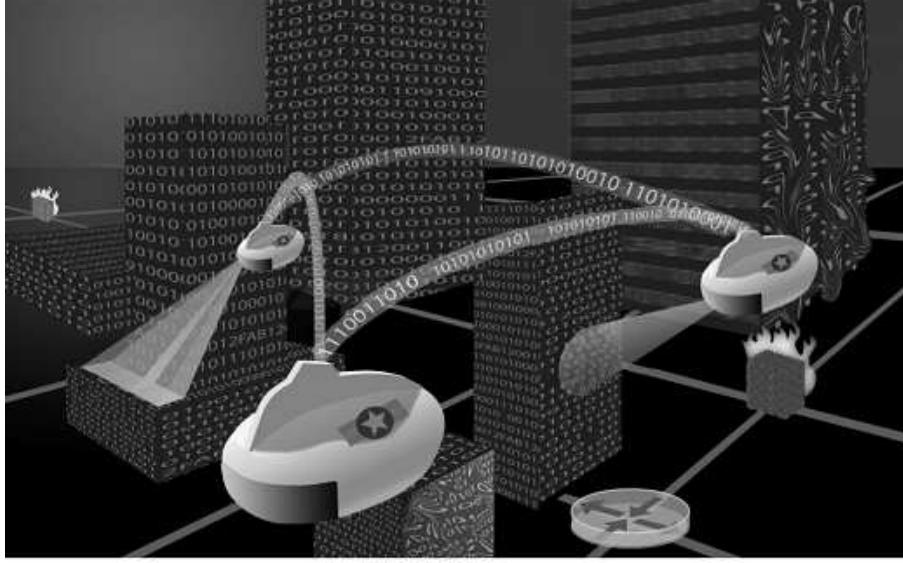


Figure 1.1: Artist's rendition of CyberCraft in Cyberspace.

These cyberspace weapon systems, CyberCraft, will be built to the CyberCraft Initiative's standards by a variety of defense contractors, much as the aircraft and satellites employed by the USAF are built to standards by contractors. The CyberCraft consist of the agent and the payloads carried by the agent, similar to a F-16 Falcon and the variety of bombs and pods attached to the F-16's hardpoints. The agent, a software or hardware construct running on a host computer, is analogous to the F-16, able to load and deploy several payloads, while the payloads are analogous

¹The term "weapon system", used in this context, does not necessarily denote an offensive capability. Cargo aircraft and communications satellites are also considered weapon systems. Specifically "An item or set of items that can be used directly by warfighters to carry out combat or combat support missions to include tactical communications systems." [2]

to the bombs, Electronic Counter-Measure (ECM) pods, or other equipment pods attached to the aircraft. While the artist's depiction in Figure 1.1 may be fanciful and more in line with the view of a computer network from a movie, the concept of employing weapon systems in Cyberspace needs to be couched to the Air Force operational leadership in terms relevant to their experiences.

CyberCraft agents will be deployed on U.S. and coalition military networks to provide commanders management of their network and a fused picture of the health of the network. CyberCraft agents are resident on host computers, and have the ability to load different payloads to accomplish different missions, from loading a policy that changes settings on the host computer to comply with a specific INFOCON (Information Operations Condition) level to reporting on network traffic received by the host computer [14].

1.1 The need for CyberCraft

CyberCraft or a similar system is needed by the Air Force to achieve *Cyberspace superiority*. As the potential threats, the number of networked information systems, and volume of network traffic all increase exponentially, the CyberCraft fleet are needed to effectively manage the Air Force networks' security posture.

Threats: With faster networks and rapid computer processing, a human's reaction time is too slow to prevent network attacks. The SQL (Standard Query Language) slammer worm infected 75,000 hosts on the internet, the majority within 10 minutes. The SQL Slammer worm instances on the internet doubled every 8.5 sec-

onds [8], which is far faster than a human can defend against. CyberCraft agents will possess some decision making capabilities to identify and defend against network attacks faster than a human could react.

Management: The ratio of network devices to personnel is quickly approaching two network devices per human. It used to be that there was one computer in a workcenter to be shared by the personnel of that work center. Today, most airmen have a computer at their desk and commanders, executive officers, and staff personnel have a computer and a Blackberry, to say nothing of the servers and infrastructure that support each base network. Management of the computers and the information produced by the computers becomes too large and daunting a task to perform manually. There are numerous automated management systems that perform tasks like loading computer patches or policy enforcement (e.g., anti-virus software), but CyberCraft aims to integrate these tasks and produce a composite picture of the network health with an associated confidence.

Trust: The fundamental research question for the CyberCraft Initiative is: *“What is required for a commander to trust a Cybercraft to autonomously defend military information systems?”*. For a commander to incorporate the CyberCraft agents into the decision making process or to give responsibility for the defense of the network to the automated decision making agents, that commander must have some level of trust in the data provided by that agent, and some level of confidence in the ability of the CyberCraft agent to successfully and accurately perform the defense mission.

For the CyberCraft fleet to be given autonomy to act without human intervention in the defense of our military networks, the CyberCraft Initiative needs to include a mechanism in the system that specifies the trust the system has in the abilities of its agents to perform an action successfully or report data accurately.

Dr Indrajit Ray and Mr Sudip Chakraborty from Colorado State University were funded by the Air Force Research Laboratory (AFRL) and the Federal Aviation Administration (FAA) to develop a new model of trust that accounts for differing degrees of trust. The trust model they developed is the Trust Vector Model, which uses a combination of factors (experience, knowledge, and peer recommendations) to generate a trust vector to represent a trust relationship in a specific context, from which a trust value can be derived [17].

1.2 Thesis Statement

This research is sponsored by the Cyber Operations branch of AFRL at Rome Labs, NY, who also sponsor the CyberCraft Initiative. As the need for trust is vital to the CyberCraft Initiative, this research examines if the Trust Vector model is applicable to integrate trust into the CyberCraft fleet. For this research, I use the definition of trust from the Trust Vector model:

Trust is defined to be the firm belief in the competence of an entity to act dependably, reliably and securely within a specific context.

Using this definition of trust, integrating trust into the CyberCraft Initiative enables the commander to have a measure of confidence that a Cyber-operation executes

as expected. The trust that an agent's sensors will accurately sample and interpret the environment imbues the data posted by the agent with an associated probability of veracity. Finally, decision engines also have a level of trust associated with their abilities to make correct decisions, so the commander can trust at some level that the CyberCraft fleet will operate in a dependable, reliable, and secure manner to protect Air Force networks.

Thesis Statement: This research examines the use and limitations of the Trust Vector model for the CyberCraft Initiative. I hypothesize that the Trust Vector model is an acceptable model for the distributed environment of the CyberCraft fleet. This research determines whether or not my hypothesis is true. Additionally, this research explores some parameters needed to fit the Trust Vector model to the CyberCraft fleet.

Some of the problems with fitting the Trust Vector model to the CyberCraft fleet are the scalability of the Trust Vector model and the use of the Trust Vector model to identify bad data. This research addresses these problem areas, and provides recommendations to the programmers of CyberCraft agents for implementation of the Trust Vector model.

Results: This research finds that a modification of the Trust Vector model can be used to provide a metric of trust in a CyberCraft agent's actions. These modifications include altering the degradation function, defining a range for the trustworthiness of events, and incorporating a transactional paradigm to the Trust Vector model,

similar to the feedback mechanism on eBay® [1]. The modifications identified are listed in Sections 3.1 and 3.2. There are several challenges left for future research, these are detailed in Section 6.2, and could lead to further modification of the Trust Vector model.

The rest of the thesis is laid out as follows:

- Chapter II covers background on the CyberCraft Initiative, the Trust Vector model, and other work on trust in distributed systems.
- Chapter III analyzes the Trust Vector model and identifies some challenges with fitting the Trust Vector model to the CyberCraft fleet. Sections 3.1 and 3.2 specifically cover paradigms and modifications of the Trust Vector model for use in the CyberCraft Initiative.
- Chapter IV discusses the motivation and high-level design of two experiments that address the value of historical data to the Trust Vector.
- Chapter V covers the specific implementation of the experiments.
- Chapter VI covers the results and analysis of the experiments.
- Chapter VII contains the conclusions drawn from the analysis and experiments.

The Future Work section identifies other problem areas and optimization issues to be addressed by future research.

II. Background

The CyberCraft Initiative is creating a set of standards that future cyberspace weapon systems are to be built to. Because of the speed of operations in cyberspace, both offensive and defensive, the CyberCraft will operate mostly autonomously. As military operations become more dependant on systems that operate over cyberspace, the fundamental research question for the CyberCraft Initiative is:

“What is required for a commander to trust a CyberCraft to autonomously defend military information systems?” [11]

For the CyberCraft fleet to be given autonomy to act without human intervention in the defense of our military networks, the CyberCraft Initiative must include a mechanism in the system that specifies the trust the system has in the abilities of its agents to perform an action successfully or report data accurately.

This chapter focuses on the background of the CyberCraft Initiative, the Trust Vector model that was commissioned by AFRL for this project, and other work in trust between distributed systems.

2.1 The CyberCraft Initiative

A CyberCraft is a system to provide command control and communications for packages that defend Air Force information systems. [11] Analogous to an aircraft flying in cyberspace, the CyberCraft can load various software packages (payloads) to enable it to accomplish its mission. Various defensive missions that a CyberCraft might be assigned include, but are not limited to:

- Insider Threat Detection
- Intrusion Detection
- Policy Enforcement, especially as a response to a change in INFOCON
 - Websurfing filters
 - Host-based firewall management
 - Unauthorized software
- Virus and Worm Detection and Remediation
- Network Vulnerability Scanning
- Reporting on host computer's health

2.1.1 Components. As mentioned in the introduction, CyberCraft have two components: Agents and Payloads. Agents can be hardware or software constructs that live on a host computer. These agents have limited functionality, as the purpose of an agent is to be the launching platform for the payloads. An agent is essentially a three-way interface between the payloads, the host Operating System (OS), and the network (through which the agent connects with the CyberCraft Command and Control (C2) structure). The agent's mission then depends on what payloads are loaded and the mission. The associated payloads may be assigned to an agent dependant on the positioning of the agent (e.g., to perform a vulnerability scan of a base network, the mission would be assigned to an agent inside the base network rather than on a different base). Agents will be built to be generic with a long service life, so the

interface between the agent and the payload, network, and host OS can incorporate changes not yet conceived of, to best allow for flexibility of the agents. Using the F-16 example, an F-16 loads air-to-air armament to fly a Counter-Air sortie, or it loads precision guided bombs for an interdiction mission. Like the F-16, a CyberCraft agent can be assigned to different missions, but unlike an F-16, the same agent can load payloads for multiple missions and accomplish these missions concurrently, or switch between missions on the fly ¹.

Payloads fall into three categories: Sensors, Decision Engines, and Effectors. Sensors sample the environment and report on the sample. Decision Engines use the data provided by the Sensors about the environment and decide how to change the environment to conform with policy. Effectors are employed by the Decision Engines to change the environment. All three payloads must incorporate trust, so that the data supplied by the Sensors can be trusted by the other agents and the warfighters to accurately represent the environment, the Decision Engines are trusted to make correct decisions on how to change the environment (and if the environment needs changing), and Effectors are trusted to successfully change the environment in a predictable manner.

As various payloads allow the computer to accomplish different tasks, it is conceivable that a network administrator uses the CyberCraft to dynamically change the

¹The author concedes that an F-16 can carry multiple payloads and be switched from one mission to another while in flight, but an F-16 is limited by the number of hardpoints on its wings and fuselage, and limitations of geography and fuel prevent the F-16 from protecting an airbase while also flying into enemy territory to drop bombs

role of the host computers by loading different payloads to accomplish the mission. For example, if the mail server is bogging down under heavy mail load, a policy triggers the response of an agent on a workstation to load a mail server payload and serve as another mail server until the mail queues had been reduced below the trigger level. Another example is if a firewall crashed, another agent loads a firewall payload to take the place of the crashed firewall, although this requires dynamic routing changes in the network. Using the CyberCraft for dynamic network management requires several policies to specify the roles a computer may take on, a hierarchy for which computers change roles first, and when to implement a change in roles. Using CyberCraft in this way would make a network more robust and help ensure continuity of operations.

Another possibility is to use the CyberCraft to capitalize on dormant computing power. The CyberCraft could store computationally intense tasks, like breaking enemy cryptography with payloads that attack a small segment of the task. Periodically, a network monitor seeks out computers that are under-utilized, e.g., most desktops in a squadron after duty hours), and load a portion of the task and the payload into the agent resident on that system, and use the dormant computer to accomplish a small bit of the task. Thus, seemingly impossible tasks requiring years of computing power can be accomplished relatively quickly by distributing the task to computers that are otherwise unused.

An example of what a fleet of CyberCraft could do is the defense of a zero-day attack worm². CyberCraft assigned to the Intrusion Detection mission encounter the worm attack and use their sensors to determine that the worm traffic is not normal and block the unknown traffic. Other sensors then analyze the traffic and produce a signature for this new attack, and distribute this signature to the other agents in the fleet. The other agents then use effector payloads to inoculate the host network against this particular worm.

2.1.2 Fundamental Problem Areas. The six Fundamental Problem Areas for the CyberCraft Initiative reflect what problems need to be solved to build an effective CyberCraft fleet.

Map and Mission Context: The CyberCraft fleet will combine data to paint a single multilayered Common Operating Picture (COP) for the Cyber-domain. The different layers of the picture correspond to different aspects of the environment. An organizational view of the network shows the network health of the different units, even if a unit's computers and network devices are geographically and logically separated. A process view shows the various network nodes and links needed to accomplish various missions. The logical view shows how each computer was logically connected to the network through various switches and routers, while the physical view shows geographical locations of the differ-

²Zero-day attack refers to a new attack against a previously unknown vulnerability. Most Anti-Virus (AV) defenses are signature-based, meaning that the AV program requires the signature of an attack to be saved in its database so the AV program can recognize and defend against the attack. A zero-day attack does not have a published signature, so it bypasses most AV defenses

ent network nodes and links. An example of how these views work together is a NIPRNET (Non-Secure Internet Protocol Router Network) switch that supports a fighter squadron. In the organizational view, the switch appears under the fighter squadron, while in the process view, the switch shows that it supports the missions of NIPRNET e-mail and websurfing for the fighter squadron, but does not affect the mission of generating sorties or mission planning (e.g., a SIPRNET(Secure Internet Protocol Routing Network) switch). The logical view shows where the switch tied into the base's NIPRNET backbone, and the physical view shows where it was located in the fighter squadron's building.

As all the views of the switch are based on data in the environment's description of the switch, the system must be able to trust the data in the description to paint an accurate picture. If a network scanner is untrustworthy and potentially compromised, then the data from that scanner must be identified and the part of the picture that is created from that data needs to identify that the data is suspect, until the data can be verified as correct or incorrect.

Figure 2.1 shows the difference between different mapping contexts. At the top, the Organizational view depicts units in the hierarchical structure of command. The middle shows the Process view of the components of the Weather Squadron for producing a weather report. The bottom shows the network connectivity for the base, identifying which buildings are connected to which switch (useful for identifying which units are affected when a backhoe digs up the fiber optic network backbone).

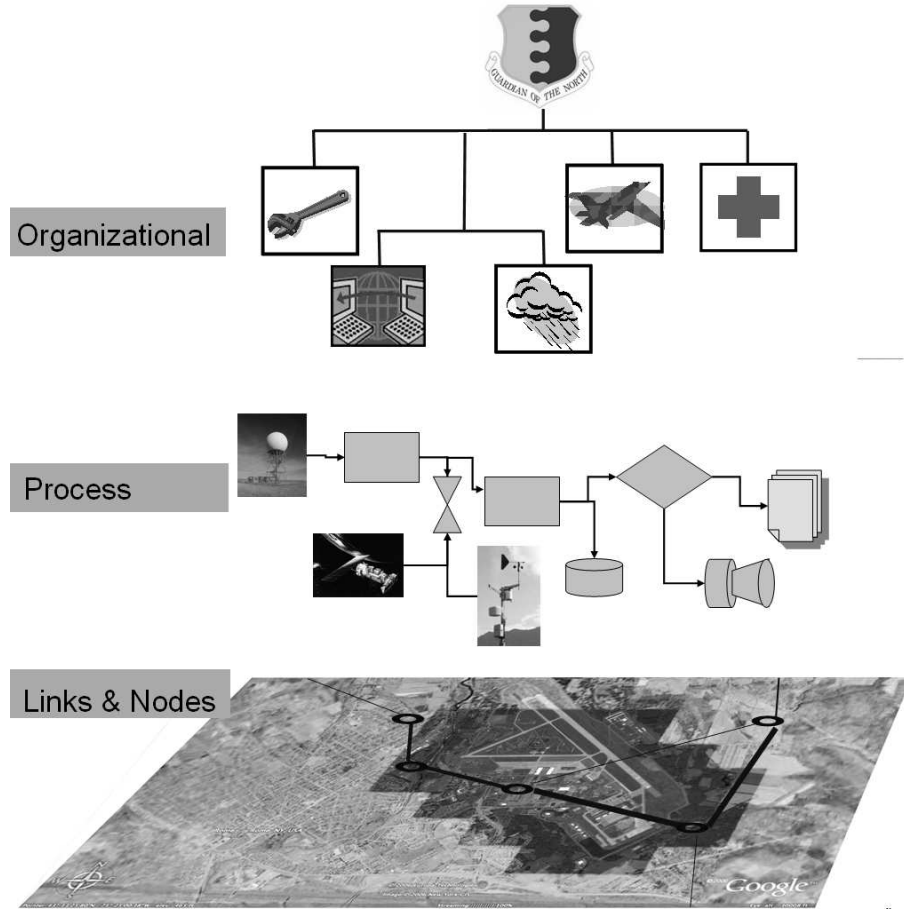


Figure 2.1: Different Mapping Contexts.

Environment Description: A common language must be developed for all sensors to describe the environment, so that data from any source is understood by all agents and payloads that access the data. The environment description is closely tied with mapping, as the system uses the description of the environment to graphically display it to the user.

The environment description must include a confidence or trust value in the data that is describing the environment. This area of research seeks to determine what information about the environment needs to be sampled and shared, to

provide enough information about the environment for a decision-maker, either human or automated, to make decisions on how to change the environment. Too much information hogs bandwidth and leads to delays in the decision making cycle as irrelevant information is weeded out, while too little information leads to incorrect assumptions about the environment and poor decisions.

An example of the resolution of the information is the reporting of the Operating System (OS) of a target machine. If the target machine is running Windows XP Service Pack 2, would a decision engine need to know all that information about the target machine, or just know that it was a Windows system?

Another challenge in the environment description language is the trust of a datum. If an agent with low trust reports that a target machine is running Windows XP and an agent with higher trust reports that the machine is running Linux, which answer is correct? Is it possible that the more trusted machine made a mistake, or that the target machine is running Windows XP on the hardware and running Linux inside a virtual machine, so both views are correct? This challenge is addressed by the introduction of a trust model to the system, but the trust model must be modified so that both views of the system are accepted as possibly correct, to take into account of the possibility of virtual machines.

CyberCraft C3 Protocols and Framework: Another research area for CyberCraft is the method for Command, Control, and Communications (C3), which details

how agents communicate between each other. Currently the prototype CyberCraft agents communicate via a distributed file store, where all agents listen to locations in the store for new policies and orders, and data posted by one agent to the store can be read by all agents. As scalability becomes a concern, this distributed store may become unwieldy, and another solution must be implemented.

Another concern is re-establishing communications if communications with an agent or group of agents has been lost. If the CyberCraft fleet has been segmented, then the different segments may continue gathering data, making decisions, and changing the environment independent of the other segments. When communications between the segments is restored, there may be conflicting data and policies between the two segments. The deconfliction of conflicting data and policies is a research area that needs to be addressed.

Formal Model and Policy: To formally prove the CyberCraft standards, a formal model must be built to describe the set of states that the CyberCraft can be in. Adding a trust model to the CyberCraft adds another set of factors to the set of possible states. The formal model of the CyberCraft must mathematically prove the state transitions of the CyberCraft so that with known inputs the output of the system is predictable. Jøsang theorized that trust in a system can be derived from one's trust in the formal modelling process of that model [12]. An application of this theory is presented in Section 3.1.1.

By formally defining the set of actions, events, and states, it is possible to predict the behavior of the system by simulating a desired course of action in the formal model of the system. Future work on the formal model may be able to incorporate a stochastic representation of the environment, so that probabilities of outside events could be modelled and the effects on the system be predicted.

Simulating a desired course of action with the CyberCraft fleet enables planners to estimate the success of an operation in the cyber-domain with an attached level of confidence. By integrating a trust mechanism, the formal model of the CyberCraft fleet can achieve a greater granularity for predictions. Rather than estimating the probability for a particular action, the values of the trust model give a level of trust in the ability of a particular agent to successfully accomplish an action. This enables planners to incorporate not only the probabilities of the payload (e.g., a certain patch has a 90% probability of being successfully installed), but also the abilities of each agent (e.g., the agent that is using the patching payload to install patches on remote machines has a known weakness in installing patches on Windows Server 2003 machines, therefore the probability of a successful installation is lower by using this agent rather than another agent).

This research is strongly focused on the application of trust towards simulation of a desired course of action. While the Trust Vector model benefits other Fundamental Problem Areas such as the Environment Description, this

research tailors the experiments to show the benefit of a trust metric to predict future system behavior.

When a trust model is integrated into the CyberCraft, the trust model must also be formally proven. The Trust Vector model provides concrete mechanisms for determining the level of trust in a relationship, so mathematically proving the Trust Vector model is easier than a trust model with less well defined evaluation mechanisms.

Self-Protection Policy: Self-Protection Policy refers to the ability of the CyberCraft fleet to conduct assured operations. This research area incorporates anti-tamper/software protection research, but also needs a mechanism to identify a compromised agent so that a compromised agent does not pollute the data used by the rest of the CyberCraft fleet. The Self-Protection Policy research area is tied to the Formal Model and Policy research area in that the Formal Model provides a prediction of good behavior, and the one self-protection method is to watch for unpredictable behavior from the local agent (self) and remote agents. The Self-Protection policy also helps to protect an agent from compromise by adding mechanisms to each agent similar to the work on the Trusted Platform Module (TPM) [21]. A trust model aids in enabling agents to identify which agent has been compromised, and adjust their interpretation of compromised agent's data appropriately.

CyberCraft Package Interfaces: CyberCraft agents need interfaces between the agent and the host OS, the agent and the network, and the payloads. This

research area is tied to the C3 Protocols and Framework, as the CyberCraft agent is an information conduit between the network (and by extension, the CyberCraft C2 structure), the host OS, and the payload. These interfaces must be simple and give agents the flexibility to load new types of payloads created after the agents have been built, as well as be able to accommodate changes to the host OS.

As the CyberCraft agents incorporate a trust model, this trust model must be able to sample the data received from the interface (OS, payload, or network), and evaluate the data received to ensure that the data has not been compromised. If the host OS or payload is compromised, it could be used to inject bad data into the agent, and if the network is compromised, the C2 structure could be spoofed and misleading commands or data could be introduced to the agent.

2.2 Trust Vector Model

In most trust models, trust between computers or software agents is defined in boolean terms; either an agent completely trusts another agent or it does not trust the other agent (covered in Section 2.3). Boolean trust is a good paradigm for authentication, as either a remote agent is or is not the entity they claim to be. In transactions between distributed entities where there could be ambiguity, such as transferring stock tips or buying items on an online auction site, a range of trust is more useful to the trusting party. If agent *Alice* has a track record of 70% good data, a user of that data (automated or human) assigns more merit to new data from *Alice*

rather than from agent *Bob* which has a 50% good data track record. The user must not give complete trust to the data from *Alice*, as historically *Alice* has provided bad data 30% of the time [16].

In order to describe a range of trust between two agents, Ray and Chakraborty developed the Trust Vector model, where trust is defined as a vector with three components, experience (past performance of remote agent), knowledge (ability of remote agent), and recommendations (from other agents regarding their trust in the remote agent) [17]. By their admission, these three components are not the only components that can be used to determine trust. The range of trust is a real and spans from complete trust (represented as +1) through no trust (0) to complete distrust (-1). Distrust differs from no trust in that distrust indicates a level of confidence that the information is incorrect rather than total uncertainty about the veracity of the information.

This Trust Vector model also incorporates a degradation function which represents that trust (or distrust) lessens over time to approach 0 or no trust. Modelling trust between humans, as time passes the trust one agent has in the other slowly degrades as the remote agent's abilities may have changed, or the data provided becomes stale.

The CyberCraft agents will operate in a world of ambiguity, where data may be partially true, and agents may vary in their ability to provide accurate answers. From a defensive standpoint, computer attacks are becoming increasingly sophisticated, not

only in the penetration and exploitation, but also in stealth and hiding the damage done. A network attack may look like legitimate mundane traffic, and legitimate traffic may appear to be a network attack to certain sensors. If an e-mail spam filter labels 60% of all messages from .gov domains as spam, even if it correctly labels 99.9% of all Viagra® e-mails as spam, the filter is not trusted as much as a similar filter that did not block the .gov e-mails. The spam filter that does block the .gov e-mails is not always incorrect, and there is benefit in using its input in determining what is spam and what is not. Trust models that do not provide a range of trust either accept or reject all recommendations from the spam filter. By using a trust model that represents trust as a range of values, the inputs from the filter are flagged with a trust value and evaluated with inputs from other filters to provide a recommendation to an automated evaluator if a specific e-mail is spam or not.

As the CyberCraft agents are deployed over a large number of systems and loaded with diverse payloads, much of the data received from the different agents about the same event may differ, and the trust model employed by the CyberCraft fleet must be able to describe the trust that an agent or automated decision maker has in the agent reporting the data. When the data about an event is evaluated, data from less reliable sources is given less weight than data from more reliable sources. If the only data about an event comes from an unreliable source, then that data may be used, but the confidence in that data is appropriately noted.

An example of this is an agent whose data has been mostly wrong, perhaps because the agent has been compromised and fed incorrect data. New data from

the agent is presumed to be wrong, and the actual data presumed to be something other than what is reported (e.g., the unreliable agent reports that a machine is not infected by a worm, but because the same agent has reported many other machines that were infected by the worm as being clean, the evaluating agent may presume that the target machine *is* infected by the worm).

2.2.1 Trust Vectors Overview. A trust vector is valid in a specific context; one agent may have multiple trust vectors for another agent in different contexts, denoting that it trusts the data from the other agent differently depending on the context of the data. The experience component is based on previous data received from the remote agent and the veracity of that data. The knowledge component is based on the knowledge one agent has about the abilities of the other agent in that context. The recommendation component is the sum of the recommendations of other agents on the trustworthiness of an agent in that context weighted by the trust the receiving agent in the recommendation context of the other agents.

Another aspect of trust that the Trust Vector model incorporates is that trust degrades over time unless constantly refreshed [17]. For example, I regularly took my car to an auto mechanic named Eddie for two years, and I built a trust relationship with Eddie in his ability to fix my car (as I kept using Eddie, the value of the trust relationship should be positive). If I do not have my car serviced for a year, and then needed to have the car repaired, I remember that I had a good trust relationship with Eddie, but my trust is not as strong as if I had Eddie change my brake fluid

a month ago (and updated the trust relationship). This is because Eddie had been good in the past, but I do not know if he is still as competent. Likewise, a negative trust relationship with Eddie approaches neutral trust over time, as Eddie may have learned some new skills since I used him last. The degradation function is covered in Section 2.2.4.

2.2.2 Components. To illustrate how the different parts and components of the Trust Vector model work together to provide a value for trust, I'll use the scenario of hiring a babysitter. While this may not sound related to computer security, the Trust Vector model can easily be applied to the building of my trust in a potential babysitter to watch my child. To start, I have multiple trust vectors with different contexts for each potential babysitter. While I have a high level of trust in a 16 year old girl's ability to watch my infant, I have a much lower level of trust in that same 16 year old to drive my sports car, and my level of trust in her recommendations of other babysitters may also be different.

Table 2.1 lists the symbology used in describing the different components of the Trust Vector model.

2.2.2.1 Experience Component. The experience component is based on the past performance of the remote agent in the given context. The experience component for the trust relationship between truster A and trustee B in context c is written as ${}_AE_B^c$. Each event where the performance of the remote agent is evaluated is given a value of trust positive (+) or trust negative (-). This collection of events

is then divided by time into intervals. The values of the events in each interval are summed to produce a single value for that interval. If the interval from t_0 to t_1 has four events, three trust positive and one trust negative, the value of that interval is +2. Each interval is weighted based on the number of intervals, n , and the position of each interval, i . The weight for each interval is calculated by the formula $w_i = \frac{i}{S}$ where $S = \frac{n(n+1)}{2}$. Figure 2.2 shows the calculation of an experience component with 6 intervals and an interval width of 4. The intervals are numbered from 1 to n (where n is the number of intervals) starting with the oldest interval.

Older intervals are weighted less than more recent intervals. The length of an interval is arbitrary, and the number of events that occur in an interval may not be

Table 2.1: Symbology used to describe the components of the Trust Vector Model.

Symbol	Meaning
$(A \xrightarrow{c} B)_t$	Trust Vector from agent A to agent B in context c
$(A \xrightarrow{c} B)_t^N$	Normalized Trust Vector from agent A to agent B in context c
$\mathbf{v}(A \xrightarrow{c} B)_t^N$	Value of the Normalized Trust Vector from agent A to agent B in context c
${}_A E_B^c$	The experience component for agent A 's trust vector to agent B in context c
${}_A K_B^c$	The knowledge component for agent A 's trust vector to agent B in context c
Ψ	A group of recommenders when building the recommendation component
${}_{\Psi} R_B^c$	The recommendation component for agent A 's trust vector to agent B in context c
w_i	The weight assigned to the i interval when calculating the experience component. $w_i = \frac{i}{S}$
S	Used to calculate w_i where $S = \frac{n(n+1)}{2}$ where n is the number of intervals
V_j	A recommendation from a recommender j In this implementation of the Trust Vector model, $V_j = \mathbf{v}(j \xrightarrow{c} B)_t^N$ for recommended agent B

	<div style="display: flex; align-items: center; justify-content: space-between;"> Older ← → Newer </div>					
Interval	1	2	3	4	5	6
Weight	$\frac{1}{21}$	$\frac{2}{21}$	$\frac{3}{21}$	$\frac{4}{21}$	$\frac{5}{21}$	$\frac{6}{21}$
Value	+1	0	-1	+4	0	0
Normalized Value	+0.25	0	-.25	+1	0	0
Weighted Value	+0.012	0.0	-.036	+0.190	0.0	0.0

Figure 2.2: Calculation of the experience component.

constant, but the weighted value of each interval is normalized between -1 and $+1$ by dividing each weighted value by the number of events in the interval. In Figure 2.2, the weight of the oldest interval is $\frac{1}{21}$ or 0.048, derived from the equation $w_i = \frac{i}{S} = \frac{1}{21}$, as $S = \frac{n(n+1)}{2} = \frac{6(7)}{2} = 21$. The normalized value is calculated by dividing the value of the interval by the interval width (i.e. the greatest possible value for the interval). In Figure 2.2, the normalized value for the first interval is $\frac{\text{value}}{\text{interval width}} = \frac{1}{4}$ or +0.25. After weighting the normalized value, the weighted value of the first interval is $0.25 \cdot 0.048 = 0.012$. Summing the weighted values of all intervals, the value of the experience component in Figure 2.2 is 0.16.

Now back to the Babysitter scenario; I've hired the babysitter many times before, and in the first few times that she has watched my daughter, I had bad (trust-negative) experiences with her. Five weeks ago, the house had toys strewn all over, the child was screaming, and her diaper was dirty. In recent weeks, my experiences with her babysitting have been good (trust-positive). Not only were the toys picked

up and my child clean and happy, but the dishes were also washed. If my intervals are weeks, then the recent weeks with good experiences count more than the weeks a month past where I had bad trust experiences with her.

In Section 3.2.1 and Section 3.2.2, I propose modifications to Ray and Chakraborty's experience component. The first modification concerns evaluating events in a range from $[-1, 1]$ rather than as only trust-positive or trust-negative. The second proposed modification suggests using variable length intervals which are set to fixed time periods instead of a fixed number of events.

2.2.2.2 Knowledge Component. The knowledge component represents the local agent's knowledge about the remote agent's abilities. The knowledge component for the trust relationship between truster A and trustee B in context c is written as ${}_AK_B^c$. An example of this is two remote agents each have a software package to scan a network for vulnerabilities. If one agent's package is known to have a high rate of false positives (or false negatives), then the value of the knowledge component for the context of scanning for vulnerabilities is less than the agent who has the more accurate scanner. As different scanners have different false positive rates for the different vulnerabilities, a more granular approach is to have a trust vector for the local agent's trust in the remote agent for detecting each vulnerability, but this leads to scalability issues with the amount of data being stored for multiple trust vectors.

Babysitter scenario: The knowledge component is where I account for the babysitter's knowledge about how to do her job. If she has taken an infant Cardio-

Pulmonary Resuscitation (CPR) class from the Red Cross and had earned her Babysitting merit badge from Girl Scouts, her knowledge component is higher than the babysitter who had neither of these qualifications. When calculating my trust vector for the babysitter in the context of driving my sports car (a 2001 Honda Prelude SH, with 200 horsepower and 5 speed manual transmission), my knowledge of her young age (and driving inexperience) and inability to drive a manual transmission gives her a low value for the knowledge component.

Ray and Chakraborty proposed the knowledge component to have two subcomponents, *direct knowledge* and *indirect knowledge*, but did not mention a method for determining the values for either. In Section 3.1.1, I propose a method for setting the initial values of the indirect knowledge subcomponent, and in Section 6.2.5 I discuss future research areas for dynamically changing the knowledge component.

2.2.2.3 Recommendation Component. The recommendations component is updated through querying other agents about their trust with the remote agent. For example, *Alice* is querying *David* about *Bob*. *David*'s recommendation is tempered by *Alice*'s trust in his ability to give an accurate recommendation. If *David* has a history of making poor recommendations, then *Alice* trusts his recommendation less than one from an agent that has a better history for accurate recommendations (*Cathy*). The recommendation component between *Alice* and *Bob* for context c is written as ${}_{\Psi}R_{Bob}^c$ where Ψ represents a group of agents with recommendations to *Alice*

about *Bob*. In Figure 2.3, Ψ includes *Cathy* and *David*. The equation for summing the recommendations is:

$${}_{\Psi}R_{Bob}^c = \frac{\sum_{j=1}^n |\mathbf{v}(Alice \xrightarrow{rec} j)_t^N| \cdot V_j}{\sum_{j=1}^n |\mathbf{v}(Alice \xrightarrow{rec} j)_t^N|}.$$

where the numerator represents the sum of recommendations weighted by the *Alice*'s trust value in *Cathy* and *David*, and the denominator represents the sum of *Alice*'s trust values to normalize the answer to between $[-1, 1]$.

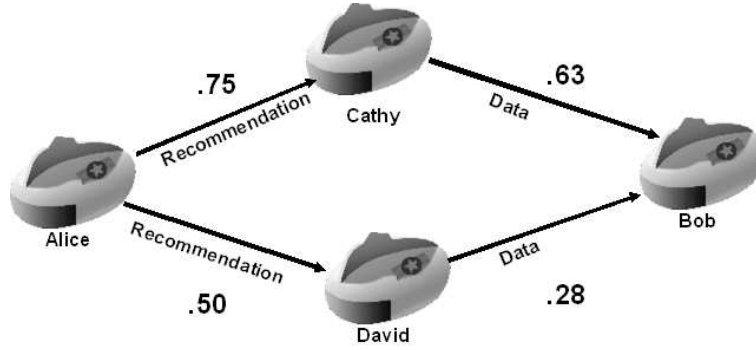


Figure 2.3: Demonstration of the recommendation component.

Figure 2.3 shows the example of *Alice* requesting recommendations from *Cathy* and *David* for *Bob* in the context of data reporting. *Cathy*'s trust value in *Bob* for data reporting is 0.63 ($\mathbf{v}(Cathy \xrightarrow{data} Bob)_t^N = 0.63$), and *David*'s trust in *Bob* is $\mathbf{v}(David \xrightarrow{data} Bob)_t^N = 0.28$. *Alice*'s trust value in *Cathy* for recommendations is 0.75 ($\mathbf{v}(Alice \xrightarrow{recommendation} Cathy)_t^N = 0.75$), and *Alice*'s trust in *David* for recommendations is $\mathbf{v}(Alice \xrightarrow{recommendation} David)_t^N = 0.50$. So as Ψ represents *Cathy* and *David*,

$${}_{\Psi}R_{Bob}^{data} = \frac{(0.75 \cdot 0.63) + (0.50 \cdot 0.28)}{0.75 + 0.50} = 0.4438.$$

The value of ${}_{\Psi}R_{Bob}^{data}$ is closer to 0.63 than to 0.28 as *Alice* has higher trust in *Cathy*'s recommendations than in *David*'s recommendations.

A problem with this equation is the use of absolute values with regards to the trust that the truster (*Alice*) has to the recommender (e.g., *David*). If *Alice*'s trust value to *David* is negative, indicating distrust with the *David*'s recommendations, then using the absolute value of the trust value negates the indication of distrust, and uses the recommendation as if the *Alice* has the level of trust in *David* as *Alice*'s level of distrust in *David*. Thus the recommendation from a highly distrusted agent is weighted more than the recommendation from a somewhat trusted agent. This equation is analyzed further in Section 6.2.6.

For the Babysitter scenario, the recommendation component is equivalent to my asking other parents that the babysitter had worked for about their trust in her ability. While getting recommendations from other parents, I use my trust in those parents' abilities to give a good recommendation to modify and normalize their recommendations. If the mother with a heart of gold and not an unkind word for anyone had given me a couple inaccurate recommendations in the past, I would value her recommendation less than the mother who had only given me accurate recommendations for other babysitters.

Table 2.2: Symbology used to describe the Trust Policy Vector.

Symbol	Meaning
W_e	The weight assigned to the experience component by the trust policy vector
W_k	The weight assigned to the knowledge component by the trust policy vector
W_r	The weight assigned to the recommendation component by the trust policy vector

2.2.3 Trust Policy Vector. Each component of a trust vector ranges in value from -1 to $+1$. To produce a single value for trust, a Trust Policy Vector is applied to the trust vector. The symbology for the Trust Policy Vector is listed in Table 2.2. The Trust Policy Vector has the same components as the trust vector, and each component of the trust policy represents the associated weight placed on the components of the trust vector, thus W_e, W_k, W_r represent the weights of the experience component, knowledge component, and recommendation component respectively. $W_e, W_k, W_r \in [0, 1]$ and $W_e + W_k + W_r = 1$. By multiplying the value of each component of the trust vector by the corresponding weight and summing the product

$$W_e \cdot_A E_B^c + W_k \cdot_A K_B^c + W_r \cdot_\Psi R_B^c \in [-1, +1]$$

a single value between -1 and $+1$ is produced, where -1 indicates total distrust of the remote agent, 0 indicates lack of both trust and distrust of the remote agent, and $+1$ indicates complete trust.

Continuing with the babysitter analogy from the previous subsection, my trust policy vector for hiring a babysitter is based on how much I valued each component. I put more weight on experience, only a little weight on knowledge, and some weight on recommendations, so my trust policy vector looks like $W_e = 0.5, W_k = 0.15, W_r = 0.35$. If I did not have many friends to give recommendations about the babysitters that I am looking to hire, and I don't have very much experience with any babysitters, then I reduce both the weight on experience and the weight on recommendations and put more weight on knowledge ($W_e = 0.25, W_k = 0.6, W_r = 0.15$).

Table 2.3: Symbology used to describe the degradation function.

Symbol	Meaning
t_i	The time at event i
t_n	The time at event n
τ	Period of decay, used to calculate Δt
Δt	The difference between time t_i and time t_n Δt is unitless, and is expressed as $\frac{t_n - t_i}{\tau}$
T_{t_i}	A Trust Relationship at time t_i
$\mathbf{v}(T_{t_i})$	The value of the Trust Relationship at time t_i
$\mathbf{v}(T_{t_n})$	The value of the Trust Relationship at time t_n
k	Rate of decay
$\mathbf{v}(T_{t_n}) = \mathbf{v}(T_{t_i})e^{-(\mathbf{v}(T_{t_i})\Delta t)^{2k}}$	Degradation Function
α	Weight corresponding to present normalized vector
β	Weight corresponding to time-dependant vector ($1 - \alpha$)
T_{t_n}	Time-dependant trust vector at time t_n Derived from $T_{t_n} = [\frac{\mathbf{v}(T_{t_n})}{3}, \frac{\mathbf{v}(T_{t_n})}{3}, \frac{\mathbf{v}(T_{t_n})}{3}]$
$\frac{v(\hat{T})}{3}$	The value of each component of the time-dependant trust vector

2.2.4 Degradation Function. As mentioned in the background, the Trust Vector model has a degradation function to calculate the current value of trust based on a past value. Figure 2.4 shows how the trust value of a trust relationship approaches 0 or no trust as time increases. Table 2.3 lists the symbology used for the degradation function.

The degradation function described by Ray and Chakraborty is

$$\mathbf{v}(T_{t_n}) = \mathbf{v}(T_{t_i})e^{-(\mathbf{v}(T_{t_i})\Delta t)^{2k}}$$

where $\mathbf{v}(T_{t_i})$ is the trust value of a trust vector (T_{t_i}) at time t_i and $\mathbf{v}(T_{t_n})$ is the degraded value at time t_n , and $\Delta t = t_n - t_i$ and k is an integer greater or equal to 1. The value for k is arbitrary and determines the rate of decline. Figure 5.5 in Chapter V shows the effects of different values of k .

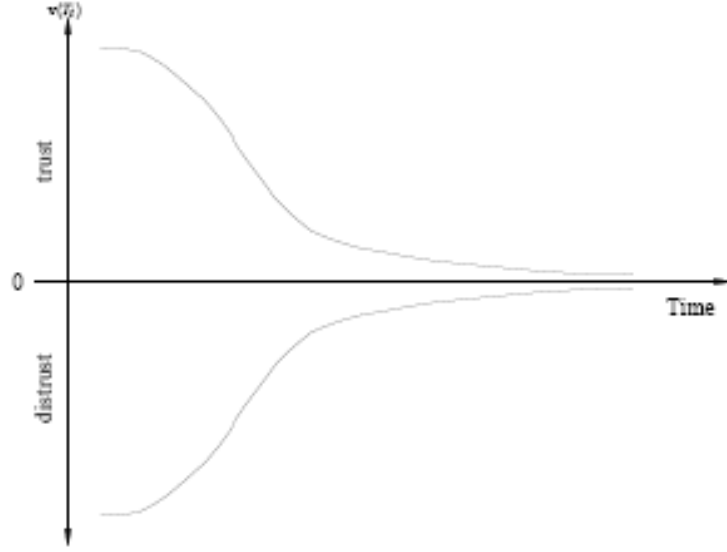


Figure 2.4: Trust degradation as time increases.

To calculate the normalized trust relationship at the present time $(A \xrightarrow{c} B)_{t_n}^N$, where the present time is represented by t_n , the values of the previous trust vector and the present trust vector are combined. The values of α and β are used to weight the present vector and the time-dependant (past) vector respectively. The trust relationship at the present time is calculated :

$$(A \xrightarrow{c} B)_{t_n}^N = \begin{cases} t_n = 0 \\ [{}_A \hat{E}_{B,A}^c \hat{K}_{B,\Psi}^c \hat{R}_B^c] \\ t_n \neq 0 \text{ and } {}_A \hat{E}_B^c = {}_A \hat{K}_B^c = {}_\Psi \hat{R}_B^c = 0 \\ [\frac{\mathbf{v}(\hat{T})}{3}, \frac{\mathbf{v}(\hat{T})}{3}, \frac{\mathbf{v}(\hat{T})}{3}] \\ t_n \neq 0 \text{ and at least one of } {}_A \hat{E}_{B,A}^c \hat{K}_{B,\Psi}^c \hat{R}_B^c \neq 0 \\ [\alpha \cdot {}_A \hat{E}_B^c + \beta \cdot \frac{\mathbf{v}(\hat{T})}{3}, \alpha \cdot {}_A \hat{K}_B^c + \beta \cdot \frac{\mathbf{v}(\hat{T})}{3}, \alpha \cdot {}_\Psi \hat{R}_B^c + \beta \cdot \frac{\mathbf{v}(\hat{T})}{3}] \end{cases}$$

As Ray and Chakraborty state in their paper [17], the speed of decay is dependant on the truster’s policy. Parameters of the degradation function can be set so that the value of a previous trust vector approaches zero in at different time periods. In Section 5.2.1, I cover my analysis of the degradation function and recommend a modification to the function.

2.2.5 Vulnerability Assessment Scenario. To illustrate how the three components of trust work together we consider a scenario where three agents are scanning the same network for vulnerabilities. This scenario assumes that all agents share and evaluate results immediately after the scan completes.

- Agent 1 has a high false positive rate when scanning for a particular vulnerability.
- Agents 2 and 3 are accurate when scanning for the same vulnerability.

All agents have two trust vectors for each of the other two agents; one vector for the remote agent’s ability to scan for the vulnerability, and another vector for the remote agent’s ability to provide accurate recommendations.

Each agent also has two trust vectors to themselves; one for their trust in their scanning ability, and the other for their ability to provide accurate recommendations. These vectors enable the agent to self identify as possibly producing bad data if the value of the trust vector to itself drops below a certain level. If Agent 1 receives recommendations from the other agents that Agent 1 is reporting systems vulnerable when they are not (due to the high false positive rate), Agent 1’s trust vector to itself

for its scanning ability decreases in value. As the value of the trust vector drops, Agent 1 has less trust in its own data, acknowledging that it may be compromised or have a problem with its scanning software.

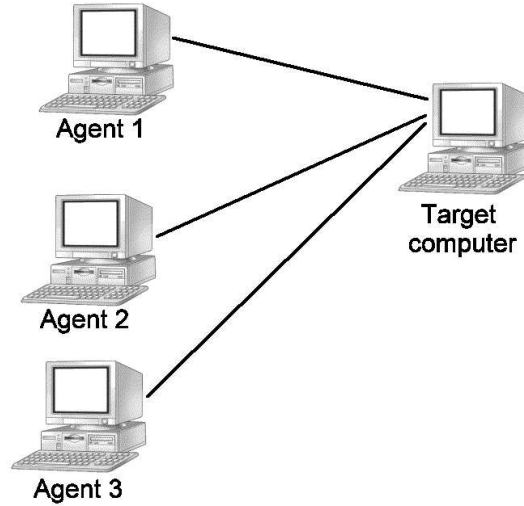


Figure 2.5: Scenario of Agents 1, 2, and 3 scanning the same computer.

While scanning the network for a vulnerability, Agents 1, 2, and 3 scan the same workstation (Figure 2.5). Agent 1 reports that the workstation is susceptible to the vulnerability, and Agents 2 and 3 report the workstation is not vulnerable.

- Agent 1 records this event as trust negative (-1) for its trust vectors for Agents 2 and 3's scanning abilities.
- Agent 2 records this event as trust negative (-1) for its trust vectors for Agent 1, and trust positive ($+1$) for Agent 3.
- Agent 3 records this event as trust negative (-1) for Agent 1, and trust positive ($+1$) for Agent 2.

Over time, as more computers are scanned, the history of events is populated and the experience components of the trust vectors for scanning can be calculated. If Agent 3 needed to make a determination as to whether or not the scanned box had the vulnerability, it modifies each result by its trust value for the agent that supplied each result. After modifying each result, Agent 3 then combines the three results to produce a single value which represents its confidence that the target machine is vulnerable or is not vulnerable.

If Agent 1 received data about a target computer that it did not scan, it is able to make a determination as to the state of that computer based off of the trust it has in the agent reporting the data.

An advantage of this approach of using multiple machines and multiple scanning techniques is that if a compromised computer is configured to hide vulnerabilities from a particular scanner, as other scanners detect the vulnerability the trust vector system notes that there is a problem with the scanner that is being spoofed (in addition to identifying the compromised computer).

2.3 Other work in trust between computers

Yahalom, Klein, and Beth build a boolean trust model [24] that models the passing of authentication data inside a network. Part of this model (YKB model) establishes a formal method for the creation of new trust relationships from existing relationships. The YKB model allowed the trust relationships to be chained to build trust relationships between entities that exist more than one degree of separation

apart (Entity A trusts B, B trusts C, C trusts D, therefore A trusts the D because C trusts D, B trusts C for C's trust to D, etc.).

This transitivity of trust can be modelled by the Trust Vector model by the use of the recommendations of agents with existing trust vectors to the remote agent to build an initial trust vector from a local agent to the remote agent. The Trust Vector model can also chain trust as long as there is some path between agents of existing trust vectors. If a path does not exist, then an new trust relationship between the local agent and remote agent can be built.

Yahalom, et al. [24] appears to be the seminal work on most formal distributed trust models. The YKB model defines trust classes which are analogous to the context of a trust relationship in the Trust Vector model, and made a distinction between *direct trust*, trust directly calculated by an agent, and *recommendation trust*, trust passed to the local agent by a recommender.

Beth, Borcharding, and Klein [5] expand on Yahalom, Klein, and Beth [24]. The expanded model (BBK model) adds degrees of trust to the previous model, based on the summation of positive experiences. The degree of trust is based upon the reliability of the remote entity, calculated $v_z(p) = 1 - \alpha^p$, where v_z is the value of the number of positive experiences p , and α is an arbitrary number between 0 and 1. The value of p ($v_z(p)$) represents the probability that the remote agent is more reliable than α . which should be large enough to safely estimate the reliability of the remote agent. Thus, if the local agent has 20 positive experiences with a remote agent and an $\alpha = 0.85$,

the reliability of the remote agent is estimated at $v_z(20) = 1 - 0.85^{20} = 0.96124$. Any negative experience causes the remote agent to become untrusted. This differs from the Trust Vector model, as a negative experience is recorded and lowers the trust in the remote agent, but the BBK model retains the boolean trust paradigm of Yahalom, Klein, and Beth's.

Jøsang does not define a trust model [12], but rather examines and defines trust in his paper. He defines trust differently between *passionate entities* (e.g., humans) and *rational entities* (e.g., computers). Trust in a passionate agent is defined as the belief that the trusted entity does not behave in a malicious (i.e. dishonest and unlawful) manner, while trust in a rational entity is defined as the belief that it resists malicious manipulation by a passionate entity. Jøsang continues with the observation that trust between humans as being based on faith, and that trust in distributed systems should be based upon knowledge, where knowledge represents the information used to determine trustworthiness. Figure 2.6 shows how the final trust in a service (*a*) is produced from current knowledge about the service (representing *direct trust* in the service), or is derived from the Trust derivation mechanism in the system about the service (*b*), which represents *derived trust*. If *a* is derived from *b*, the *derived trust* becomes explicit and becomes *direct trust*. In Figure 2.6, the trust in the system (*c*) is derived from the formal verification of the model (*d*), which is based on knowledge about the formal verification process (*e*).

In the Trust Vector model, *derived trust* is the trust calculated by the Trust Vector system, and *direct trust* is the current trust value after it has been derived.

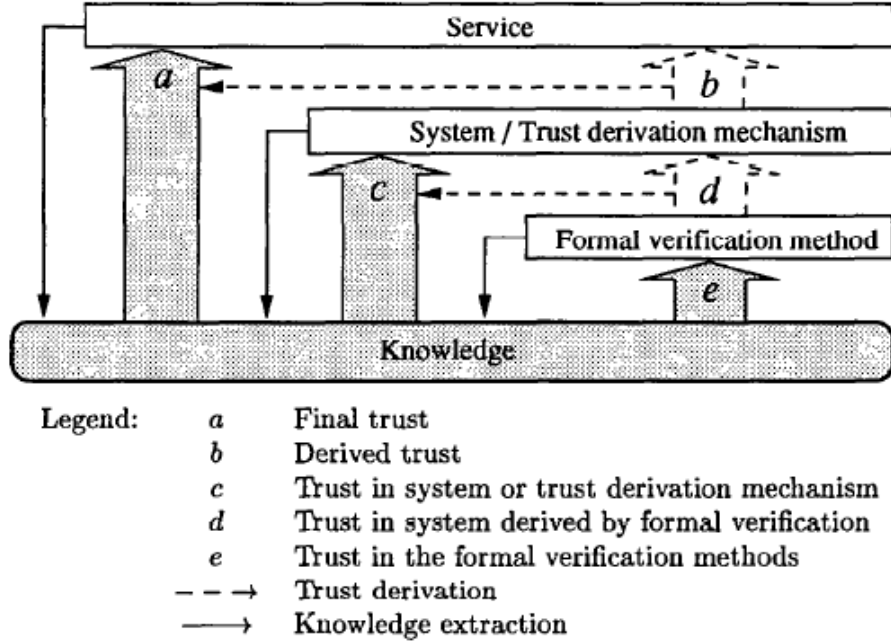


Figure 2.6: Trust Based on Knowledge (Jøsang).

The Trust Vector model updates the trust value as new data is received and evaluated, so a trust relationship is constantly being derived. *Direct trust* only occurs if no new information had been received since *derived trust* was calculated, and this departure from Jøsang’s paper occurs because the Trust Vector model does not use boolean trust, but rather the range of trust. In a boolean trust model, a service or entity is trusted until that service performs an action that results in the loss of trust, while a non-boolean model needs to constantly determine the level of trust attributable to a service or entity.

Jøsang completes his paper with an evaluation on the differences between security and reliability. Between *passionate entities* security is derived from the trust in the other entity’s benevolence, and reliability is derived from trust in the other entity’s skill and experience. To trust a *rational entity* for security, the rational entity’s

strength against manipulation is used, while trust in reliability is based on knowledge of the rational entity's capability for continuous operation. Using these definitions, the CyberCraft Initiative seeks to capture the trust in rational entities reliability, but substitute predictability for continuous operation as the basis for reliability.

Purser's Graphical Model of trust introduces a range of trust (low, medium, and high), and specifies that trust depended on the context of the relationship [15]. Purser's model also incorporates an associated risk to the trust relationship that indicates the potential damage caused by a breach of trust. Finally, the model uses a transitivity attribute to indicate if the trust relationship held from one actor to another actor can be passed on to a third actor. Figure 2.7 shows an example of the trust from a patient to their doctor. The trust extends from the patient to the doctor, the context is medical advice, the level of trust is high, the associated risk with a breach of that trust is high, and the trust is transitive (e.g., a patient trusts their doctor, the doctor trusts a surgeon, therefore if the trust between the doctor and surgeon is transitive, the patient trusts the surgeon (in the context of medical matters)).

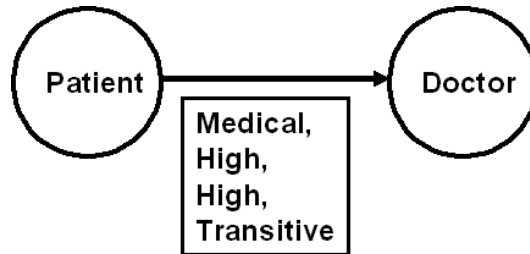


Figure 2.7: Purser's Graphical Model of Trust.

This model (which uses unidirectional arrows to indicate the trust relationship from the truster to the trustee), is good for humans planning the trust between agents or entities, as the resulting graph is visually intuitive to capture a large number of trust relationships between multiple entities, but is not as useful to translate into code representing trust between computers. The lack of trust mechanics and the ambiguous levels of trust are difficult to manage trust between computers. The transitivity of trust is somewhat analogous to the recommendations between agents in the Trust Vector model.

Abdul-Rahman and Hailes' trust model [3, 4] is similar in many respects to the Trust Vector model. Both Abdul-Rahman/Hailes' model (A-R/H model) and the Trust Vector model incorporate different contexts for trust. Trust is based on experiences and recommendations (although Trust Vectors add a knowledge component as well), and trust is not directly transitive (a recommendation is modified by the trust in the recommender). The A-R/H model also incorporates degrees of untrustworthiness, but trust values are discrete (Very Untrustworthy, Untrustworthy, Trustworthy, and Very Trustworthy in [4], and Distrust, Ignorance, Minimal, Average, Good, and Complete in [3]), rather than the continuous range used by the Trust Vector Model. The A-R/H model constructs a message protocol for passing recommendations between agents, as well as equations for determining the trust value of a recommendation path [3].

Xiong and Liu [23] proposed a trust model for peer-to-peer (P2P) eCommerce transactions called PeerTrust. PeerTrust provides a range of trust relative to a peer's

performance in transactions based on feedback received from the other peers involved. The PeerTrust model combines five factors to produce a value for the trustworthiness of a peer. These factors are:

Feedback in Terms of Amount of Satisfaction: This is based on the feedback from the other peer in an eCommerce transaction. Feedback is the basis for most trust models in a P2P eCommerce environment.

Number of Transactions: As the name suggests, this is the number of transactions performed by the rated peer in a given time period. This is included to provide an average feedback score per transaction, as a skewed distribution of transactions may not fairly capture the trustworthiness of a peer (e.g., the peer who receives 6 negative feedbacks over 12 transactions is less trustworthy than the peer who receives 7 negative feedbacks over 1000 transactions). eBay® uses a form of this when presenting the percentage of positive feedback an account has received [1].

Credibility of Feedback: This factor captures potential threats of dishonest feedback. If a peer is providing negative feedback on positive transactions for a malicious motive, a credibility factor captures the lack of trustworthiness of the misbehaving peer's feedback, and future feedback from this peer is adjusted accordingly.

Transaction Context Factor (TCF): This factor is arbitrary and is set depending on the community modelled, indicating which transactions are more valuable

(and therefore, which deserve more weight when calculating trust). In an online auction community, the TCF could be the dollar amount of the transaction, showing that a peer could be trustworthy for lower dollar amounts, but shady for larger dollar amounts. Thus the feedback from an expensive transaction weighs more than the transactions for inexpensive transactions.

Community Context Factor: The community context is also arbitrary, and models factors important to a specific community. Xiong and Liu use the example of a business community that uses historical ratings (trust ratings from outside the captured time period) to help evaluate the peer based on consistent behavior. Another example is a file sharing community (like pre-lawsuit Napster), where the number of transactions that add new files to the community is used as the Community Context Factor. This models the community's desire to add more music into the community, rather than have the same files swapped between peers.

The inclusion of these five factors are to expand the common model of feedback used in online auction sites that use just the feedback from the other peer in a transaction. The models used by online auctions assume that all feedback is honest, and that no deception or ulterior motive is behind the feedback ratings. Other motivation for this model was the perceived lack of incentives for one peer to rate another, the lack of decrementing the weight placed on older ratings, the lack of context for the ratings.

This model can be used to provide several of the same abilities as the Trust Vector model. The decay of older values can be modelled by the Community Context Factor. The trust vectors for recommendations can be modelled by the Credibility of Feedback factor. Different contexts in the Trust Vector model (e.g., trust in the babysitter to watch my child vs. trust in the babysitter in driving my car) can be modelled by two different instantiations of the PeerTrust model. The advantage that the Trust Vector model has over PeerTrust for the CyberCraft Initiative is the Trust Vector model's experience component and degradation function handles the weighting of recent data over older data more gracefully. Xiong and Liu's PeerTrust model could be expanded to include the experience weighting function of the Trust Vector model's experience component as well as the degradation function for historical data. These incorporations should also include analysis of how much historical data must be kept.

Ray, Chakraborty, and Ray have built a model trust management framework as an extension of the Trust Vector model [18]. In this framework, VTrust, information about trust relationships is stored in a trust database. This framework uses a modification of SQL named TrustQL to query the database for trust information. This research did not address the use of VTrust, instead each agent storing its trust data locally. In Chapter VI, the use and possible future research of VTrust in the CyberCraft fleet is expounded upon.

Chakraborty and Ray also extend a variation of the Trust Vector model to the control of privacy in online transactions [7]. The p-Trust model is similar to the Trust Vector model, with the following differences:

- A trust relationship in p-Trust is defined by four components, *interactions*, *properties*, *reputation*, and *recommendation*. *Properties* and *reputation* were subcomponents of the knowledge component of the Trust Vector model (direct knowledge and indirect knowledge respectively).
- The concept of a null relationship or null value for a component is added. Values of components and trust relationships now range from $[-1, +1] \cup \{\perp\}$, with $\{\perp\}$ representing a null value.
- The recommendation component of p-Trust (${}_R REC_B^c$) is analogous to the recommendation component of the Trust Vector model (${}_\Psi R_B^c$). However, the equation to normalize the p-Trust recommendation component does not use the absolute value of the truster's trust in the recommending agent (${}_R REC_B^c = \frac{\sum_{j=1}^n \mathbf{v}(A^c_j) \cdot V_j}{\sum_{j=1}^n \mathbf{v}(A^c_j)}$). This may indicate that the equation for the Trust Vector model needs to be updated.

The p-Trust model also categorizes violations of privacy into seven categories (*Confidentiality Breach*, *Integrity Breach*, *Information Exploitation*, *Personal Space Violation*, *Pretexting/Identity Theft*, *Anonymity Violation*, and *Linkability*). Categorizing potential violations of trust in the Trust Vector model could help to control the damage if a violation occurs.

III. Trust Vector Architecture

To use the Trust Vector model for the CyberCraft Initiative, the architecture of the model must first be analyzed to determine how it should be integrated into the CyberCraft design. This research focuses on what settings and modifications are needed to use the Trust Vector model to integrate a trust metric to distributed agents such as the CyberCraft fleet. This research also found areas for expansion of the Trust Vector model, borrowing from other trust models that Section 2.3 covers. Most of these expansions add to the Trust Vector model without changing the model, but some of the modifications replace the implementation of a function or change how certain data is interpreted and stored.

This chapter covers how to fit the architecture of the Trust Vector model to the CyberCraft model. Section 3.1 covers existing properties of the Trust Vector model and how they can be fit to the CyberCraft model. Section 3.2 covers the recommended settings and modifications to the Trust Vector model.

3.1 Properties of the Trust Vector model

This section discusses new ways to use the existing properties of the Trust Vector model. Some of the following sections cover ideas from other trust models that are not explicitly mentioned in the Trust Vector model.

3.1.1 Knowledge Component. Ray and Chakraborty [17] discuss the two subcomponents of the knowledge component, *direct knowledge* and *indirect knowl-*

edge, but do not mention a mechanism for initially setting the values. As Jøsang wrote in his paper [12], trust in a service can be derived from trust in the system, which can be derived from the trust of the formal verification of the system. From this, assigning higher values to the knowledge component of trust relations to the formally modeled components of the CyberCraft is one way to assign an initial value to the knowledge component. All CyberCraft agents will undergo a formal verification, but not all payloads will. Thus a trust relationship in the context of using a formally verified payload has a higher knowledge component than a relationship for a non-verified payload. Not all payloads in the CyberCraft fleet need to be formally verified. One of the advantages to the CyberCraft Initiative is it can use a payload that is generated quickly and cheaply, therefore giving the CyberCraft fleet agility in countering new threats. But a payload that is rushed into production may not always behave as a formally verified payload, thus, the lower trust inherent in a lower knowledge component.

Dynamically changing the knowledge component is an area of future research, and is covered in Section 6.2.5.

3.1.2 Transaction Paradigm. Ray and Chakraborty do not mention the paradigm the Trust Vector model must be used in. One paradigm uses multiple agents to sample the same aspect of the environment (e.g., scanning the same computer for vulnerabilities) and immediately exchange their results, and evaluate the results of the remote agents. This *synchronous or co-stimulation paradigm* works well for

the Trust Vector model, as each participating agent obtains a large amount of data for the other participating agents and a robust experience component can be built. This synchronous paradigm works well for determining which agents interpret the environment differently than the other agents. Systems like an Air Traffic Control (ATC) network or the multiple guidance computers on the Space Shuttle operate in synchronous paradigm.

When CyberCraft is implemented, it is unlikely that multiple agents can immediately sample the same aspect of the environment to evaluate the data produced by an agent. Currently CyberCraft communicate via the Pastry [19] implementation of a Distributed Hash Table (DHT), which is used as the Information Store, where all data (environment descriptions, policies, payloads, etc.) resides. Information gathered by agents is posted to the Information Store and is accessible by any other agent. This may lead to scalability and communication issues later, but this example is based on the premise that a version of the Information Store exists in the final implementation of CyberCraft. The data a CyberCraft agent posts to the Information Store may not be immediately evaluated by another agent, and when the data is accessed, the accessing agent may not have the means to verify the data. This *transactional paradigm* is closer to the way an eCommerce environment such as eBay® works.

Xiong and Liu's paper [23] discuss the use of trust for an eCommerce environment, an example being an online auction community (e.g., eBay®). In an online auction, each purchase of an item represents two transactions, the buyer receiving the goods purchased and the seller being paid by the buyer. After each transaction, the

recipient of the transaction is able to rate their satisfaction with the transaction. If a buyer purchased a compact disc (CD) and the CD was scratched, the buyer would have a low satisfaction with the transaction. The satisfaction value of each transaction builds the trust that a buyer has with a seller that future transactions will be enjoyable. A history of low satisfaction with transactions indicates that future transactions are more likely to also disappoint.

Extending the *transactional paradigm* of an online auction to CyberCraft, data produced by an agent (*Alice*) is posted for other agents to access. When another agent (*Bob*) accesses the data, the data is evaluated for usefulness and veracity, and the evaluation is saved as a trust event in *Bob's* experience with the *Alice*.

In the following scenario, a transaction is the receiving of data (reading data from the Information Store) or the receiving of recommendations. The validity of the data or recommendation influences the evaluation of the satisfaction of the transaction. Good data or good recommendations lead to a good feedback score. The feedback score is then interpreted as the value of that event in the experience component. Figures 3.1 to 3.5 show how the Trust Vector model is used in a transactional paradigm.

In Figure 3.1, *Alice* gathers data about the environment and posts the data to the Information Store. All data posted to the Information Store retain the identity of the author and the time posted to the store for attribution purposes. The data for the trust relationships, including recommendations, is stored in the Information

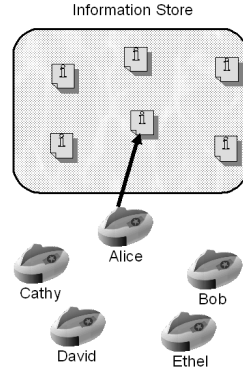


Figure 3.1: Transactional Paradigm: *Alice* posts data to the CyberCraft Information Store.

Store, and therefore an agent's recommendations are available to other agents. *Bob*

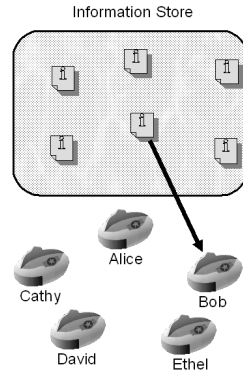


Figure 3.2: Transactional Paradigm: *Bob* retrieves *Alice*'s data from the Information Store.

queries the Information Store for information about the environment and retrieves the data posted by *Alice* (Figure 3.2). *Bob* has an existing trust relationship with *Alice*, and applies the trust value of that relationship to the analysis of *Alice*'s data. *Bob* then evaluates the usefulness of *Alice*'s data and records the evaluation as an event in the trust relationship between *Bob* and *Alice*. The value of the trust relationship is possibly altered positively or negatively depending on the results of the evaluation of the data.

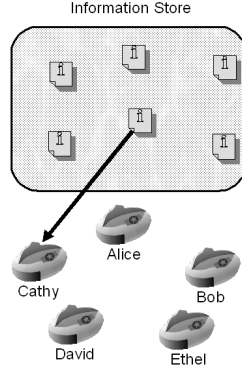


Figure 3.3: Transactional Paradigm: *Cathy* also retrieves *Alice*'s Data from the Information Store.

Cathy also retrieves *Alice*'s data (Figure 3.3), but does not have a trust relationship with *Alice*. Therefore, to build a trust relationship, *Cathy* queries other agents (*Bob*, *David*, and *Ethel*) for recommendations about *Alice*'s trustworthiness (Figure 3.4). This is accomplished by retrieving *Bob*, *David*, and *Ethel*'s recommendations about *Alice* from the Information Store.

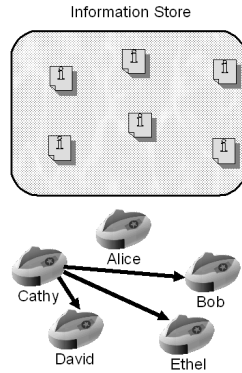


Figure 3.4: Transactional Paradigm: *Cathy* queries other agents for recommendations about *Alice*.

In Figure 3.5, *Cathy* receives recommendations about *Alice*'s trustworthiness from *Bob*, *David*, and *Ethel*. These recommendations are then modified by *Cathy*'s trust in each recommender to provide a good recommendation. If *David* did not have

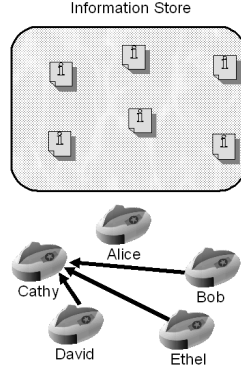


Figure 3.5: Transactional Paradigm: *Bob*, *David*, and *Ethel* send recommendations to *Cathy*.

a trust relationship with *Alice*, *David*'s recommendation is 0, or trust neutral with regard to *Alice*.

As the values of trust vectors are influenced by the Trust Policy vector, it may be useful to dynamically determine the Trust Policy vector. In the above example, when *Cathy* is requesting recommendations about *Alice*, it makes sense to put more weight on a recommendation from someone who has many transactions with *Alice* (and therefore a more populated experience component) than a recommendation from an agent with only a few transactions with *Alice*. Similarly, if *Bob* has a long history of transactions with *Alice*, *Bob* may want to put more weight in his experience with *Alice*, while an agent with few transactions with *Alice* may want to put less weight on their experiences with *Alice* and more weight on recommendations to determine the trust value.

3.1.2.1 Challenges with the Transactional Paradigm. One problem

with using the *transactional paradigm* over the *synchronous paradigm* is an agent

that is producing data that few other agents read only has trust relationships with those few other agents. This can lead to problems in chaining trust (see Section 3.1.3), as well as a greater possibility of bad recommendations. An example of this shortcoming is an agent (*Alice*) that posts data to the store that is generally only read by a one other agent (*Bob*). If a new agent (*Cathy*) reads the data and needs to create a trust relationship with *Alice*, the only other agent with a trust relationship is *Bob*. If *Cathy* did not have a trust relationship with *Bob* for recommendations, *Cathy* needs to create the trust relationship to *Bob* for recommendations by querying which other agents have a trust relationship to *Bob* for recommendations, and building the trust relationship to *Bob* from recommendations from these other agents (assuming that *Cathy* does not then need to build new trust relationships for recommendations to the agents that have existing trust relations to *Bob*, etc.). Once *Cathy* establishes a trust relationship to *Bob* for recommendations, because *Bob* is the only one with a trust relationship to *Alice* for data, *Cathy*'s new trust relationship to *Alice* is based entirely on *Bob*'s recommendation.

If two agents, *Cathy* and *David* are both compromised, and *Cathy* posts bad data to the Information Store and *David* gives positive feedback to *Cathy*, unless other agents also evaluated *Cathy*'s bad data, *David*'s trust in *Cathy* is presumed to be genuine, and *Cathy*'s data is trusted when it should not be.

Another challenge with the *transactional paradigm* is that without immediate evaluation of data, the trust in the agent may not accurately reflect the veracity of the data. If an agent (*Alice*) posts accurate data at t_0 , then posts bad data from

$t_{0.5}$ on, and the data is not read until t_1 , what trust does another agent (*Bob*) assign to the data? Does *Bob* remember the trust he had with *Alice* at t_0 ? If so, should he apply that knowledge to *Alice*'s data? *Bob* knows that *Alice* has recently posted bad data, but he does not know when *Alice* started posting bad data. If *Alice* is not compromised, but the accurate data that she posted at t_0 is now wrong (e.g., *Alice* detected a vulnerability in a target computer at t_0 , but the computer was patched at t_1 , and *Alice*'s data is accessed at t_2), should trust in *Alice* decrease?

3.1.2.2 Hysteresis. One method of addressing the challenges above is to integrate a Hysteresis mechanism. Hysteresis operates like a float in a toilet's tank, it adds water to the tank when the water is low and shuts off the water when the tank is close to full. A Hysteresis mechanism searches the Information Store for data that does not have enough feedback. If the mechanism finds a datum without enough feedback, the mechanism directs other agents to access the data and evaluate the datum to see if the datum is accurate.

The Hysteresis mechanism could also search the information data base for data that is too old and data whose trust is too low and direct agents to sample the environment to corroborate the data in the Store.

3.1.2.3 Dynamic Trust Policy. The scenario in Section 3.1.2 mentions using different Trust Policy Vectors given different circumstances. A Dynamic Trust Policy is one that assigns different weights to the Trust Policy Vector under different circumstances. If a piece of data comes from an agent without many recommenda-

tions, then the Trust Policy Vector for evaluating trust in that agent shifts to put more emphasis on experience and knowledge.

3.1.3 Transitivity of Trust. Yahalom, Klein, and Beth’s paper [24] discussed the transitivity of trust, as *Alice* trusts *Bob*, *Bob* trusts *Cathy*, therefore *Alice* trusts *Cathy*. The Trust Vector system can be used to transfer or chain trust as well, but there is a low watermarking type issue with it. Using *Alice*, *Bob*, and *Cathy* again, if *Alice* trusts *Bob* at 0.8 for recommendations, and *Bob* recommends *Cathy* to *Alice*, then the most *Alice* trusts *Cathy* initially is 0.8, as that is her trust to *Bob* for recommendations, and that is only if *Alice*’s new trust to *Cathy* is based solely on recommendations and if *Bob* trusts *Cathy* completely. If trust passed through a recommendation chain of 5 people who all trust the other links at 0.9, the final trust is 0.59049, again if new trust were solely based on recommendations.

Still, it is possible to use the Trust Vector model to create a new trust relationship by passing recommendations from one agent to another, and that new relationship increases as trust-positive events occur.

3.2 Recommended Settings and Modifications

Here is a list of recommended settings and modifications to the Trust Vector model for integration into the CyberCraft Initiative.

3.2.1 Experience Component: Event Values. A modification to Ray and Chakraborty’s model that this research explores is using a range of values from

-1 to $+1$ to describe a trust event, rather than defining all events as either trust-negative(-1) or trust-positive ($+1$). This modification allows greater granularity to describe an event. If the babysitter in the previous chapter didn't change the baby's dirty diaper and allowed the baby to scream, this experience is rated lower than an experience where the baby's diaper is dirty but the baby is happy. Both experiences are trust-negative events, but a dirty diaper and unhappy baby may be recorded as -0.8 , while a dirty diaper but happy baby may be recorded as -0.3 . This is useful for calculating the experience component for the Trust Vector for Recommendations, as it is difficult to discretely determine if a recommendation is Trust Positive or Trust Negative. Instead, this research took the absolute difference between the recommendation from the remote agent and the current trust from the local agent and used that as a measurement as to how trust positive a recommendation was (lower differences were more Trust Positive). This is shown in Figure 3.6.

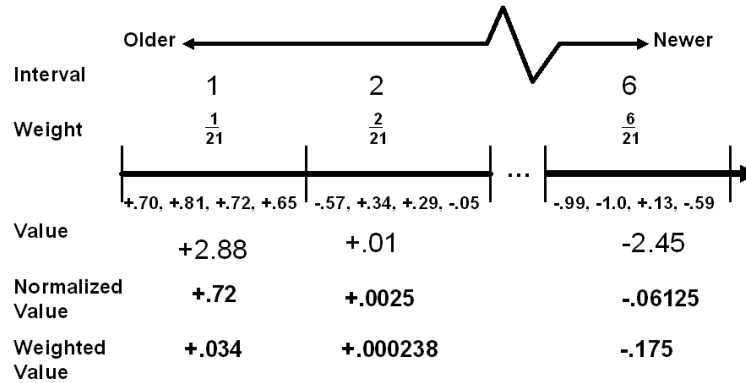


Figure 3.6: Modification of the experience component: Event values.

3.2.2 Experience Component: Intervals. This proposed modification extends the modification to the event values discussed in Section 3.2.1.

My experiments focused on Intervals with fixed widths. As an agent received events from another agent, these intervals were populated without regard to how quickly the events occurred. A more realistic use of these Intervals is to set the intervals to a fixed time, and have a variable number of events per interval. To determine the weighted value of each interval, sum up the events per interval, divide the sum by the number of events (to normalize the value), multiply the quotient by the weight of the interval, and sum the weighted values of the intervals. Figure 3.7 is a visual representation of the proposed modification.

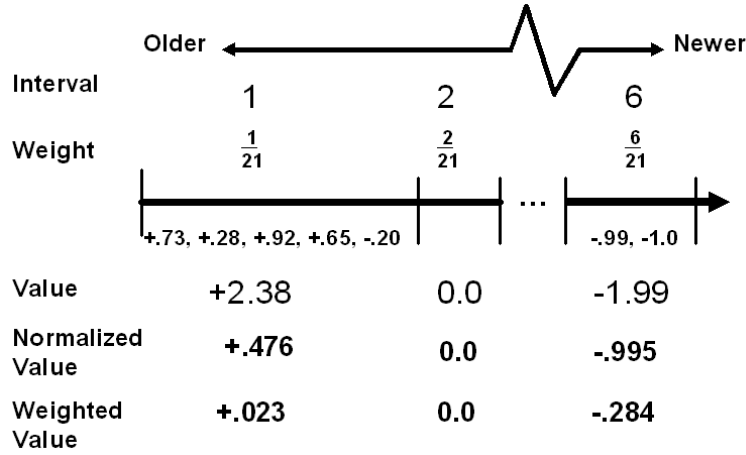


Figure 3.7: Modification of the experience component: Fixed time intervals.

In Figure 3.7, Interval 1 has 5 events occur during that time period, Interval 2 has no events, and Interval 6 has 2 events, but each the value of each interval is normalized between $[-1, 1]$ before being weighted. One potential problem with this approach is that intervals without events are assumed to be trust neutral, and skews the experience component towards trust neutral. A better approach is to ignore the

intervals without events and normalize the weights of the intervals with events. This is a good candidate for a future experiment (Section 6.2.4).

Finally, from testing results, it is recommended that no more than 10 intervals are kept for the experience component (Section 6.1.1) When 10 intervals are kept, the 10th interval provides 1.5% of the value of the experience component.

3.2.3 Integration of Entities. I propose that entities be introduced into the Trust Vector model as any service or object that can be the target of a trust relationship. Abdul-Rahman and Hailes' paper [3] described static entities as those entities that cannot execute the Recommendation Protocol, such as printers or disk volumes. An agent is defined as an entity that can engage in the Recommendation Protocol, so in terms of the Trust Vector system, an agent is an entity that holds a trust relationship with other entities, and can therefore pass the value of the trust relationship to another agent in a recommendation (the truster in a relationship must be an agent, but the trustee need not be).

Ray and Chakraborty do not elaborate on the existence of non-agent entities, so this proposal is not as much a modification to their work, but rather an expansion.

3.2.4 Associated Risk of a Trust Relationship. An idea from Purser's Graphical model was to include the associated risk (severity of consequences) if trust was breached [15]. The CyberCraft Initiative may benefit from the inclusion of a risk field to trust relationships, and require a higher trust threshold for relationships that have

a higher associated risk. Patching systems against a worm has a higher risk than installing Windows Media Player 10.

3.2.5 Degradation Function. Experiment II explores the degradation function, and modifications to the degradation function are listed in Section 5.2.1. Experimental results conclude that the *period of decay* is best set no less than 1.5 times the expected time between trust calculations (Section 6.1.2).

IV. Experimental Methodology

The goal of the two experiments is to identify the limits of the utility of historical data in calculating the value of a trust relationship. This supports the overall goal of the research by determining how much data must be kept, which then supports the scalability of the model.

This chapter outlines the two experiments, then describes a scenario to give the experiments a real world context, and concludes with the broad methodology used to build the experiments. Specific implementation details for each experiment are covered in subsections of the methodology. This methodology is based on the NIST handbook [13].

4.1 Experiment I: Limits of Historical Data for Calculating the Experience Component

As described in Section 2.2.2.1, the experience component is composed of intervals which contain a number of events. When calculating the value of the experience component, the values of the events in each interval are summed to produce a value for the interval. The values of the intervals are then normalized between $[-1, 1]$, weighted to put more emphasis on recent performance (the new intervals are weighted heavier than the older intervals), and the weighted interval values are summed to produce the value of the experience component. As the older intervals are weighted less than newer intervals, at what point do the older intervals become insignificant?

4.1.1 Problem Statement. As storage is a factor with multiple trust vectors across a large number of agents, how much history must be stored? At what point does the weight of the interval diminish the value of the interval to the point that it is not worth storing the data?

4.1.2 Hypothesis. There is a threshold for the utility of historical data for the experience component of the Trust Vector model, after which the cost of storing the data far outweighs the benefit provided by the data.

4.1.3 Motivation. As the CyberCraft fleet is expected to encompass 1,000,000+ agents, the scalability of data storage becomes critical.

4.2 Experiment II: Utility of Degradation Function

Previous trust values are used in conjunction with current trust values to produce a composite trust score. If an agent had produced poor data in the past but is now producing what appears to be good data, the current trust level is moderated with historical knowledge of the previous performance. Another scenario has an agent that has not been heard from for a while, but is now providing data again (e.g., the reporting machine was turned off and has since been rebooted). The degraded trust value is used to estimate the value of the new data, as it is imprudent to accept the data at the previous trust level as it is unknown what has happened to that agent in the interim.

4.2.1 Problem Statement. In Section 2.2.4, the degradation function is described. How fast should trust degrade? Two factors of the degradation function control how fast trust degrades (the *period of decay* (τ) and *rate of decay* (k)), so where should these factors be set? This experiment only tests the settings of the *period of decay* (τ) in relation to the exchange rate, as the effects of the *rate of decay* (k) can be seen in Figure 5.5.

4.2.2 Hypothesis. The *period of decay* must be set to at least twice the expected time between trust calculations. If the period of decay is shorter than that, the degradation function decays the previously calculated trust value too rapidly to be useful.

4.2.3 Motivation. For the CyberCraft to integrate a degradation of trust, the degradation function must be understood. If the degradation function decays previous trust values too rapidly, the previous trust values approach trust neutrality when combined with the current trust value. If the previous trust value has decayed to trust neutrality, then the cost of storing the trust data, calculating the decayed value of the trust relationship, and calculating the combined trust value exceeds the benefit of knowing the previous trust value.

4.3 OS Fingerprinting Scenario

To illustrate the effect of keeping different amounts of historical experience data on the trust value, the following deterministic scenario was developed. A group of

five agents are fingerprinting a single target machine that runs Windows and can run Linux as a virtual machine. When the target machine switches to the virtual machine, one agent detects the Linux OS and reports the target machine as running Linux, while the other agents continue to identify the target machine as running Win XP. This leads to discrepancies between the data of the agents, and the trust between the agent that detected Linux and the other agents that detect Win XP decreases, until all agents report the same OS again (when the target machine closes the virtual machine running Linux).

4.3.1 Experiment I: Scenario. In Experiment I, the five agents scan the target machine every minute and exchange new recommendations every five minutes. Four of the agents do not detect the virtual machine and continue to report the OS as being Windows XP while the fifth agent detects the virtual machine and reports that the OS is Linux (while Linux is running). This could be due to a different scanning package that the fifth machine is using that is able to detect the virtual machine, or the other agents are able to see past the virtual machine to the host OS. The difference in the reports from the fifth machine and the other four machines leads to the trust between the four machines that detect Windows XP and the fifth machine to diminish, while the trust between the four machines that detect Windows XP increases. This scenario shows the difference in trust when the target machine switches from Windows to the Linux virtual machine at 9:00 AM and switches back

to Windows at 9:30 AM (when all five machines again agree upon the detected OS, leading to increase in trust amongst the agents).

4.3.2 Experiment II: Scenario. In Experiment II, the degradation function is included in calculating the value of the trust relationships. The five agents scan the target machine every minute, but in this experiment, the agents exchange new recommendations every four minutes. Unlike the scenario for Experiment I, all agents agree on the data, so the difference in results stem from only the level of the *period of decay*.

Every time recommendations are exchanged, an agent evaluates the new recommendations, calculate current trust, and combine the current trust with the time-dependant trust (degraded value of the last recorded trust) to determine the value of the trust relationship at the current time. The current value of the trust relationship is then recorded as the time-dependant trust. During minutes when data is not exchanged, the value of the trust relationship is calculated using the time-dependant trust, degraded appropriately.

4.4 System Boundaries

The system under test is a CyberCraft agent implementing the Trust Vector model, as shown in Figure 4.1. The experiments measure the behavior of the Trust Vector model in the *synchronous paradigm*, so multiple agents read data from the same target or set of targets, exchange data about the target, evaluate data received

from remote agents immediately upon receipt, and produce a set of trust values for each agent (including a set of trust vectors for oneself). Further testing may be needed of the behavior of the Trust Vector model in the *transactional paradigm*.

Each agent contains a Data Reading component, a set of Trust Vectors, a Trust Evaluation Algorithm, and communication between the agents to exchange data and trust values.

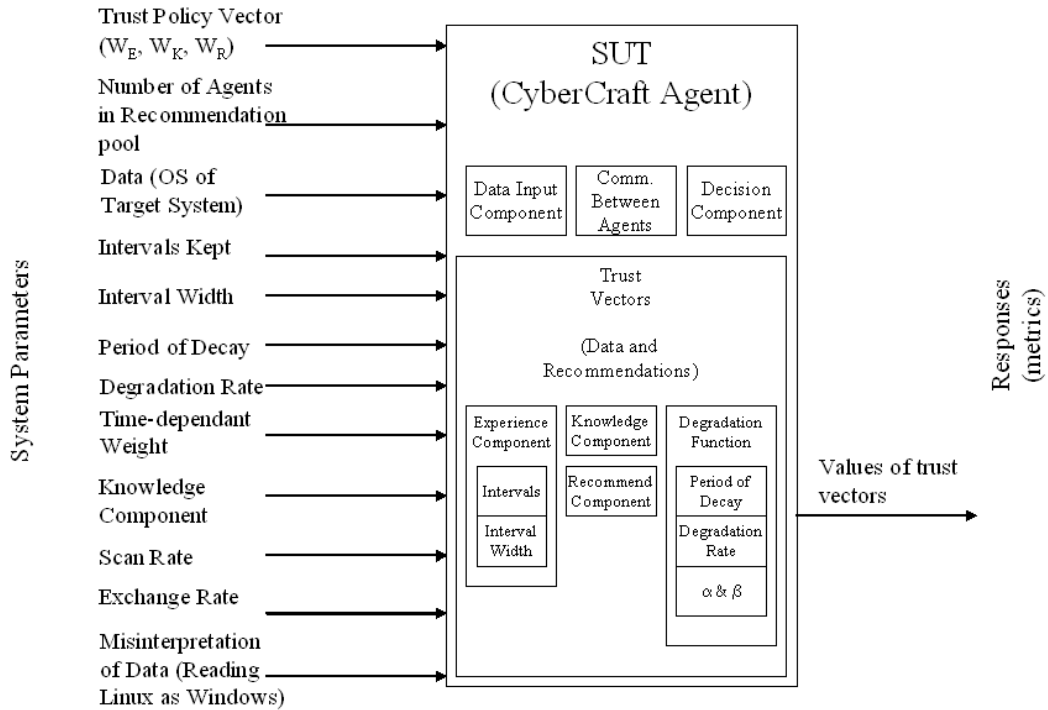


Figure 4.1: System Under Test: Implementation of the Trust Vector model.

4.4.1 Communications component. The current implementation of CyberCraft agents use a distributed file system (Pastry) to communicate. All communication between the agents is through posting files to the distributed file system. Future iterations of the CyberCraft may discard the Pastry framework and agents may com-

municate through message passing instead, but these experiments are testing the Trust Vector model under the implementation of CyberCraft. These experiments assume that the communication channel between all agents is robust; therefore the failure of communication between agents is not tested in this experiment.

4.4.2 Data Input component. The Data Reading component simulates a sensor payload for the CyberCraft agent, gathering data about the environment and outputting a value for the state of the environment. In the Operating System (OS) fingerprinting scenario, the payload is a network scanner, reporting on the OS of the target machine. To simulate a real network scanner, the Data Reading component may misinterpret data gathered from the environment (e.g., a target machine is running WinXP on the hardware, but is also running Linux in a virtual machine. One network scanner identifies the target machine as running WinXP, while another scanner identifies the target machine as running Linux). This misinterpretation is not necessarily wrong, but causes conflict between reporting agents.

For this research, the misinterpretation of data is deterministically controlled by the system parameter Misinterpretation of Data, and could be different for each agent.

4.4.3 Decision component. Future experiments should focus on the ability of a CyberCraft agent to make a correct decision given ambiguous information. The Decision component gathers various inputs and the associated trust, and makes a

decision with an associated confidence as to the true state of the environment. This future research is covered in Section 6.2.9.

4.4.4 Trust Vectors. The set of Trust Vectors contain values for the three components of the trust vectors between agents for the two contexts, reading data and recommendations.

The experience component has two variables: the number of intervals kept, and the number of events per interval. These variables are controlled by the system parameters: Intervals and Interval Width.

The recommendation component is influenced by the number of agents in the recommendation pool.

The degradation function has four variables: Period of decay, degradation rate, α and β . The first two are controlled by the system parameters of the same name, β is controlled by the parameter Time-dependant Weight, and α is calculated from β ($\alpha = 1 - \beta$). The period of decay (τ) is the amount of time used to calculate the difference in time (Δt) between timestamps. If the period of decay is 5 minutes, then the Δt between $t_i = 6$ and $t_n = 10$ is $0.8 \left(\frac{t_n - t_i}{T} \right)$, but if the period of decay was increased to 10 minutes, the Δt between 6 and 10 becomes 0.4.

The degradation rate is represented as k in the degradation function, which controls how fast a vector's value degrades with the passage of time. Higher values of k initially degrade the value slower, and then have a more rapid decay (at about $\Delta t = 0.4 \cdot | \mathbf{v} T_{t_i} |$).

The variables α and β control how much weight is assigned to the decayed value of a previous vector when combining the value of the previous vector and the value of the present vector.

4.4.5 Experiment I: System Boundaries. Figure 4.2 shows the system boundaries of Experiment I, with the parameters held constant on the top of the system diagram, the factor that is changed in the experiment on the left of the system, and the metric of the experiment displayed on the right. The system boundaries for Experiment I remain the same as detailed in Section 4.4, with the exception of the lack of the degradation function.

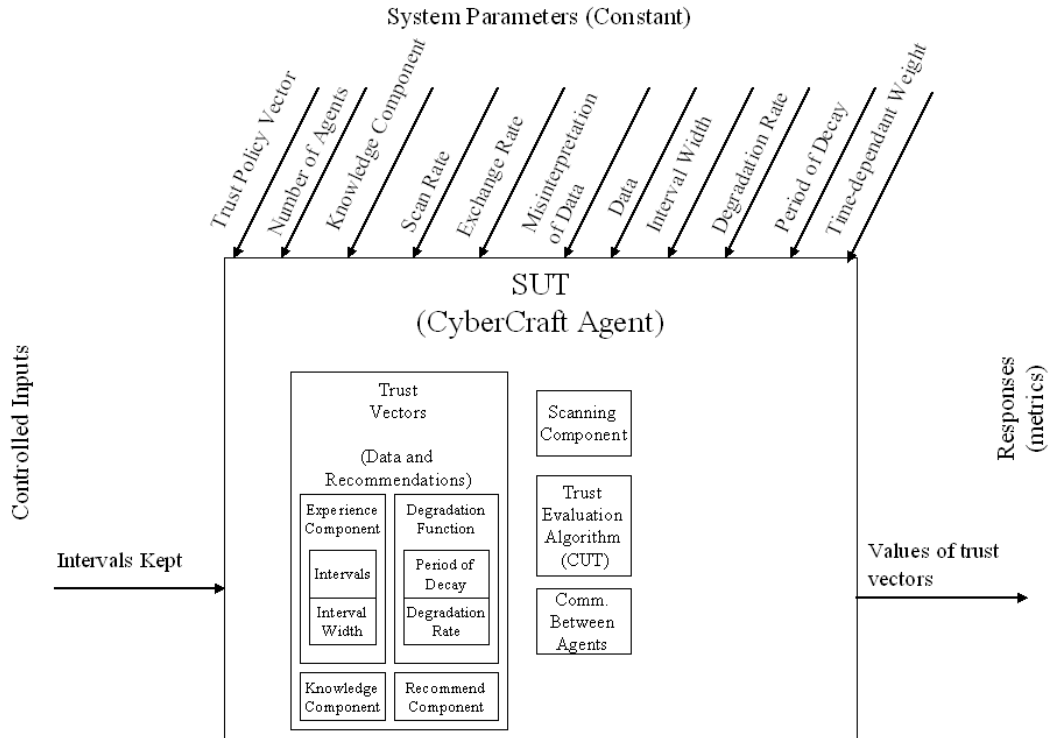


Figure 4.2: Experiment I Design.

4.4.6 *Experiment II: System Boundaries.* Figure 4.3 shows the system boundaries of Experiment II, again with the parameters held constant on the top of the system diagram, and the factor that is changed in the experiment on the left of the system.

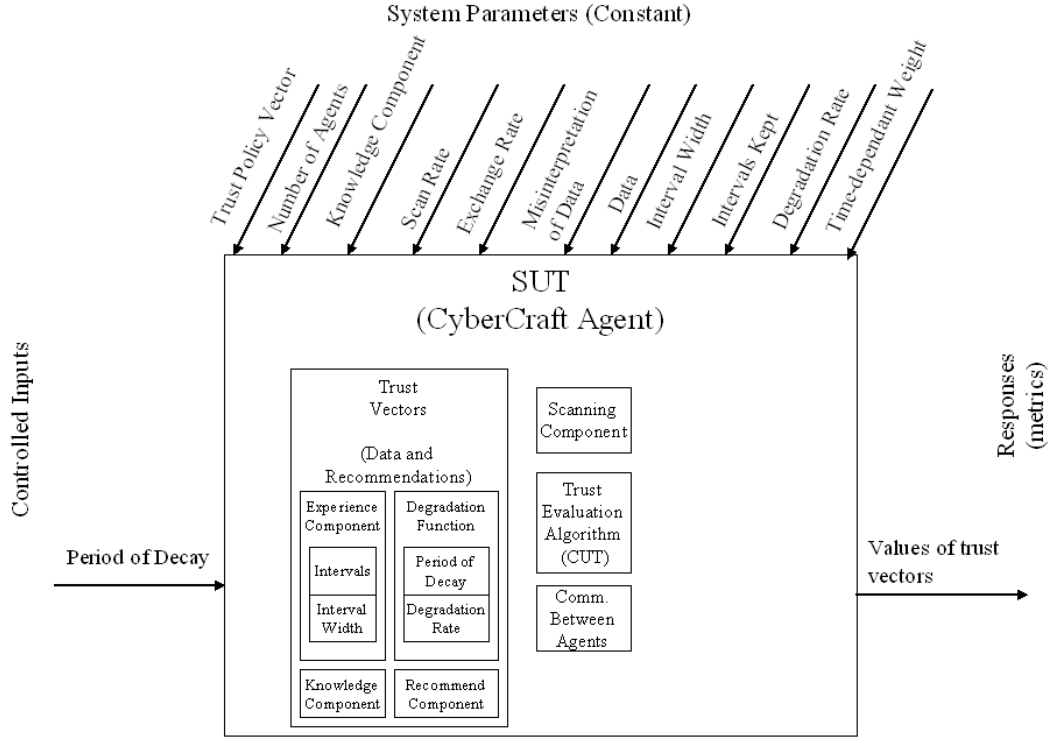


Figure 4.3: Experiment II Design.

4.5 System Services

The CyberCraft agent provides two services, the state of the environment as it sees it (data output) and recommendations for the abilities of other agents (Trust Recommendations). The Trust Recommendations are equal to the value of the normalized trust vector after modification by the Trust Policy Vector and the degradation function. In both experiments, the data output is the OS Fingerprinting.

4.6 *Workload*

The workload for the system is the number of events observed. For both experiments, an event is the simulation of a fingerprinting of a target machine's operating system.

Depending on such parameters as the number of agents in the system, the exchange rate, and the scan rate (rate at which events occur), an event can trigger multiple messages exchanging data reports and recommendations between the agents, which could pose a challenge to the operation of the system. It is hypothesized that the experiments do not create enough message traffic to slow the system from normal operations.

4.6.1 Experiment I: Workload. This experiment contained 180 events. The events from event #60 and event #120 were interpreted differently by the agents (one agent fingerprints the OS as Linux while the other machines fingerprint the OS as WinXP).

4.6.2 Experiment II: Workload. This experiment contained 180 events. All agents fingerprint the OS as Linux when the virtual machine is running Linux.

4.7 *Metrics*

Both experiments use one of the values of a Trust Relationship as the metric to determine the magnitude of difference the factors for the experiment cause in the trust value. Specifically, the value of the Trust Relationship from Agent 0 to Agent

4 for OS Fingerprinting (as opposed to Recommendations) is used as the metric for both experiments.

4.8 Parameters

The parameters for the experiments are the attributes of the environment or the implementation of the Trust Vector model that affect the output. All parameters for the system are shown in Table 4.1.

Both experiments used the following parameter settings:

- Trust Policy Vector: $\{W_e = 0.4, W_k = 0.25, W_r = 0.35\}$
- Number of agents = 5
- Interval Width = 4 events
- Knowledge Component = 1
- Scan Rate = Once per event (Once a minute)

Experiment I used the following settings for the other parameters:

- Data =
 - 60 events (representing 1 hour from 0800 to 0859) where the data is Win XP.
 - 60 events where the data is Linux.
 - 60 events where the data where the data is Win XP. again

Table 4.1: Parameters of the Trust Vector Model.

Symbol	Meaning
Trust Policy Vector	Weights assigned to each component when calculating the Trust Value. Covered in Section 2.2.3.
Number of Agents	The number of agents exchanging data about the same environment. Also the number of agents in the recommendation pools.
Data	Information about the environment each agent is scanning. In these experiments, the OS of the target system.
Intervals Kept	Number of Intervals kept to calculate Experience Component. Section 2.2.2.1
Interval width	Number of events per interval for Experience Component
Period of decay	The time period between t_0 and t_1 in the degradation function. Section 2.2.4
Degradation Rate	Integer value of k in the degradation function The effect of k on the function is shown in Figure 5.5
Time-dependant weight	β in combining function
Knowledge component	The value for the Knowledge Component This value is kept constant for all the experiments in this research. Future research will concentrate on dynamically modifying this value in as result of performance
Scan rate	How often does an agent acquire data about the environment. In these experiments, how often does the agent scan the target machine's OS.
Exchange rate	How frequently does an agent send its recommendations to the other agents. This implementation of the model uses a "push" method for exchange of recommendations, rather than a "pull" method, where recommendations are exchanges when polled for.
Misinterpretation of data	This parameter represents the tenancy of an agent to misinterpret the results of a scan as a different OS, leading to discrepancies of data between agents.

- Intervals Kept is the factor for this experiment. The levels for this factor are specified in Section 4.10.
- Exchange Rate = Every 5 minutes

- Misinterpretation of data: One agent detects the Linux OS from the virtual machine, The other agents do not detect the virtual machine and only detect Win XP.
- Degradation rate: Not tested
- Period of decay: Not tested
- Time-dependant Weight: Not tested

Experiment II used the following settings for the other parameters:

- Data = 180 events (representing 3 hours from 0800 to 1100) where the data is Linux
- Intervals Kept = 10 intervals
- Exchange Rate = Every 4 minutes
- Misinterpretation of data: Not tested (All agents detect the Linux OS)
- Degradation Rate: $k = 2$
- The period of decay is the factor for this experiment. The levels for this factor are specified in Section 4.11.
- Time-dependant Weight: $0.\bar{3}$

4.9 Factors

The factors of the experiments are the parameters that are varied between runs.

The factors for each experiment are detailed below.

4.10 Experiment I: Factors

Experiment I varied the number of intervals kept by the experience component. The levels were 7, 8, 9, 10, and 11 intervals kept.

4.11 Experiment II: Factors

Experiment II varied the period of decay. The levels for the experiment were:

- 4 minutes ($\tau = 1$ times the exchange rate)
- 6 minutes ($\tau = 1.5$ times the exchange rate)
- 8 minutes ($\tau = 2$ times the exchange rate)
- 12 minutes ($\tau = 3$ times the exchange rate)

4.12 Evaluation Technique

Both experiments evaluate the effects of the factor on the metric, the value of the trust relationship from Agent 0 to Agent 4 in the context of OS Fingerprinting is plotted for every event (Figures 5.2 & 5.6). The plots for each run are compared against each other to determine the effect that the change of the factor level had on the trust value from Agent 0 to Agent 4 in the context of data reporting.

4.13 Experimental Design

Both experiments test four levels of one factor in a deterministic system. As the test is deterministic, only one run per factor level is needed. Thus, four runs, one at each factor level, are performed.

V. Analysis and Results

This chapter covers the analysis of each tested component and the results of each experiment. Experiment I examined the utility of older intervals when calculating the experience component. Experiment II focused on exploration of the degradation function, to include defining the period of decay and determining the effect of the degradation rate. The analysis and results for each experiment are listed in the respective sections for the experiments.

5.1 Experiment I: Limits of Historical Data for Calculating the Experience Component

5.1.1 Analysis. To determine how much historical data must be stored, we first analyzed how much the historical data contributes to the current trust value. Historical data is used to calculate the value of the experience component (which in turn is used to calculate the trust value of the trust vector), and in the current value of a previous trust vector.

As stated in Section 2.2.2.1, the experience component is calculated by weighting the values of the intervals of the event history and summing the products of the weights and the values of the intervals. Table 5.1 shows the weight associated with the oldest interval kept for a given number of intervals.

As shown in Table 5.1, if 10 intervals of data are stored for the calculation of the experience component, the oldest interval counts for less than 2% of the total value of the experience component.

Table 5.1: Decreasing value of Oldest Interval.

Number of Intervals	$S = \frac{n(n+1)}{2}$	Weight of Oldest Interval
1	1	100%
2	3	33.3%
3	6	16.7%
4	10	10.0%
5	15	6.7%
6	21	4.8%
7	28	3.6%
8	36	2.8%
9	45	2.2%
10	55	1.8%
11	66	1.5%
12	78	1.3%

The following scenario illustrates the difference between keeping 11 intervals of historical data vs. only keeping 10 intervals (Figure 5.1). If the events in the 11th interval are all trust-negative, and all events since the oldest interval were trust-positive, the normalized value of interval 1 (the oldest interval) is -1 and the normalized values of the rest of the intervals are $+1$. The difference between summing the weighted values for the 11 intervals versus the ten most recent intervals is 0.03 (twice the weight of the oldest interval). An alternative scenario is that the oldest interval's value is still -1 and the rest of the intervals' values are 0. The difference in this case is 0.015, equivalent to the weight of the oldest interval. The maximum difference between the keeping 11 intervals and keeping 10 intervals occurs when the eleventh interval's value was one extreme and the rest of the intervals were the other extreme.

From this, the maximum difference between keeping x intervals and keeping $x-1$ intervals is twice the weight of the oldest interval when x intervals are kept. From

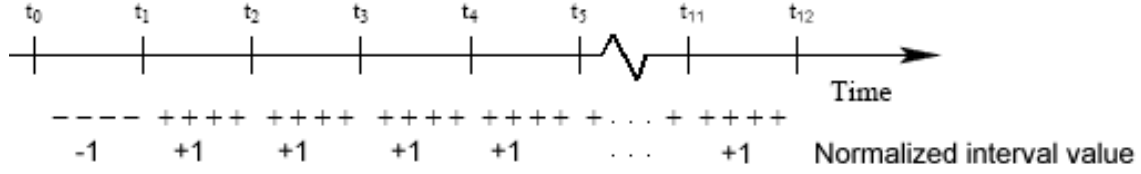


Figure 5.1: Calculation of experience component.

Table 5.1, the weight at 8 intervals is 2.8%, so to keep 7 intervals versus 8 intervals could lead to a 5.6% difference in the value of the experience component. Each application that uses the trust vector system have different needs for the granularity of trust, and Table 5.1 contains the information to determine the number of intervals needed to meet the level of granularity. The experience component is only one part of the trust vector system, and the impact it has on determining trust is governed by the trust policy. If the trust policy is set where the experience component is 50% of the trust value, then the 5.6% difference in the value of the experience component leads to a 2.8% difference in the trust value.

5.1.2 Experiment I: Results. Figure 5.2 shows the differences in the Trust Value of the agents reporting Windows OS with regard to the agent reporting the Linux virtual machine ^{1 2}. The graph shows that as the number of intervals kept decreases, the Trust Value changes faster and drops farther when discrepancies occur, as there is less historical data to counterbalance current data. The difference between 8 intervals kept and 9 intervals kept is greater than the difference between 10 intervals

¹This scenario was based on only the current calculated Trust Values, and not previously calculated Trust Values with the degradation function.

²This scenario also only concentrates on the discrepancy between reports between agents. A real world application of OS fingerprinting must incorporate some method to account for Virtual Machines or dual boot machines. This is covered in the Future Work section.

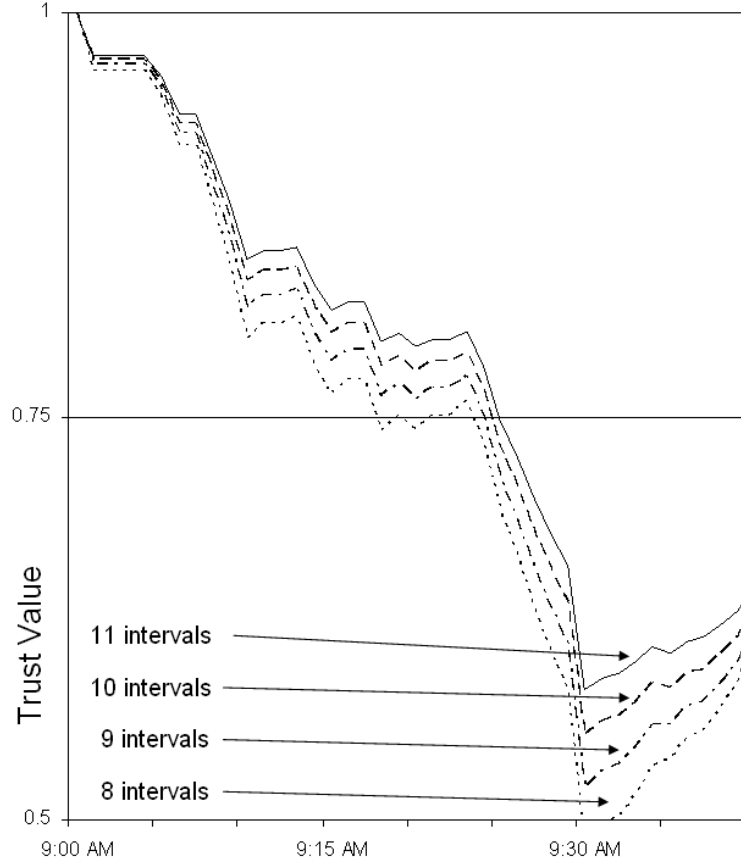


Figure 5.2: Trust Value for different experience intervals kept.

kept and 11 intervals kept (shown in Table 5.2). The maximum difference of the Trust Values between runs with different number of intervals kept is consistent with the mathematical analysis in Table 1. The maximum difference between keeping 9 and 10 intervals of data in the scenario was 0.0321, which is slightly more than 1.6% of the range of Trust Values (-1 to 1). The 1.6% difference in trust values is less than the 1.8% of the experience component listed as the value of the oldest interval when 10 intervals are kept (which is to be expected, as the experience component only accounted for 35% of the Trust Value).

In the application of the Trust Vector Model, the decision of how many intervals are to be kept depends on the trade-off between storage of each interval and the need for granularity of the Trust Values. The storage cost of each interval can be calculated by $E \times B \times C \times A$ where

- E = Events per Interval
- B = Bytes per event
- C = Number of contexts
- A = Number of agents

This scenario uses four events per interval, but this is arbitrary. If the time period of the interval is fixed but the occurrence of a trust event is stochastic, then the number of events per interval could vary depending on the rate of events generated.

The Trust Vector Model proposed by Ray and Chakraborty uses boolean values for each trust event, so every event is either trust-positive or trust-negative, but there are many contexts where a range of values for the trust event is more applicable. One example of this is the recommendations. The modified Trust Vector Model that is used in this research uses floating point values between -1 and 1 to represent each event for recommendations, as it makes more sense to represent an agent's trust in a recommendation as the absolute difference between that recommendation and the agent's trust value for the same context rather than a boolean value of agreeing with the recommendation and not agreeing with the recommendation. This scenario used one byte for every event.

This scenario only used two contexts, OS Fingerprinting and Recommendations, but it is conceivable that scanning a network for vulnerabilities could have a different context for every vulnerability scanned (or the ability of a network scanner could be abstracted into a single context).

This scenario used five agents, so each agent keeps one Trust Vector per context for all other agents plus a Trust Vector to itself for each context. Thus each interval costs 40 bytes to store per agent ($4 \times 1 \times 2 \times 5 = 40$).

Table 5.2: Decreasing value of Oldest Interval.

Number of Intervals	Maximum difference of Trust Values
8 to 9	0.0357
9 to 10	0.0321
10 to 11	0.0287

5.1.3 Results. Our results support our hypothesis in that as data ages, the benefit provided by the older data is diminished to the point where the contribution to the current trust level is so small that keeping the data is not worth the storage cost. We do not attempt to identify a specific point to discard data, but provide a model which can provide recommendations based on the implementation of the Trust Vector model

5.2 *Experiment II: Utility of Degradation Function*

5.2.1 Analysis. In analyzing Ray and Chakraborty’s equation, we saw that higher initial values of trust degrade faster than lower initial values, which is counter intuitive (Figure 5.3). Inverting the $v(T_{t_i})$ term (trust value at time t_i) from the

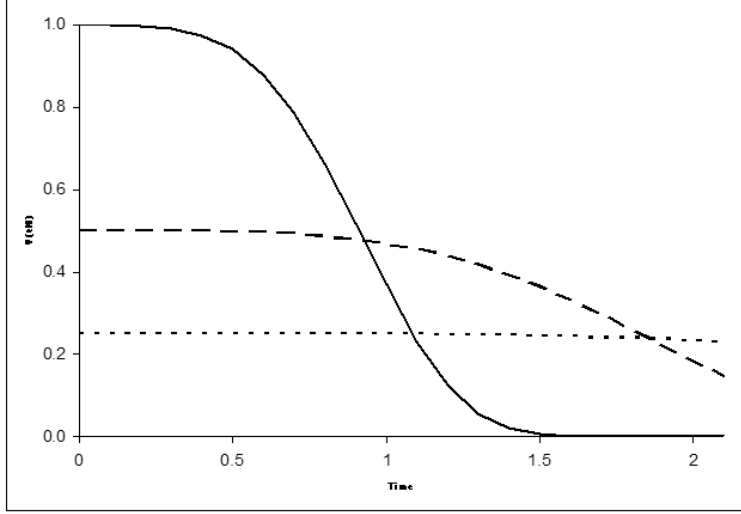


Figure 5.3: Higher values degrade faster using $\mathbf{v}(T_{t_n}) = \mathbf{v}(T_{t_i})e^{-(v(T_{t_i})\Delta t)^{2k}}$.

exponent of e changes the equation to $\mathbf{v}(T_{t_n}) = \mathbf{v}(T_{t_i})e^{-(v(T_{t_i})^{-1}\Delta t)^{2k}}$, which degrades all trust values at an equal rate as shown in Figure 5.4 (k is arbitrarily set to 2 to match figure 2).

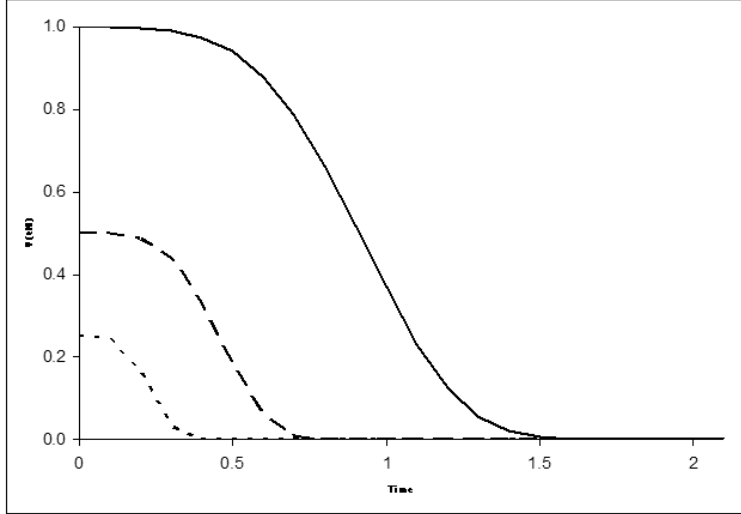


Figure 5.4: All values degrade at an equal rate using $\mathbf{v}(T_{t_n}) = \mathbf{v}(T_{t_i})e^{-(v(T_{t_i})^{-1}\Delta t)^{2k}}$.

Figure 5.5 shows the degradation of the trust values based on different decay rates. In the equation $\mathbf{v}(T_{t_n}) = \mathbf{v}(T_{t_i})e^{-(v(T_{t_i})^{-1}\Delta t)^{2k}}$, k is an arbitrary value for the

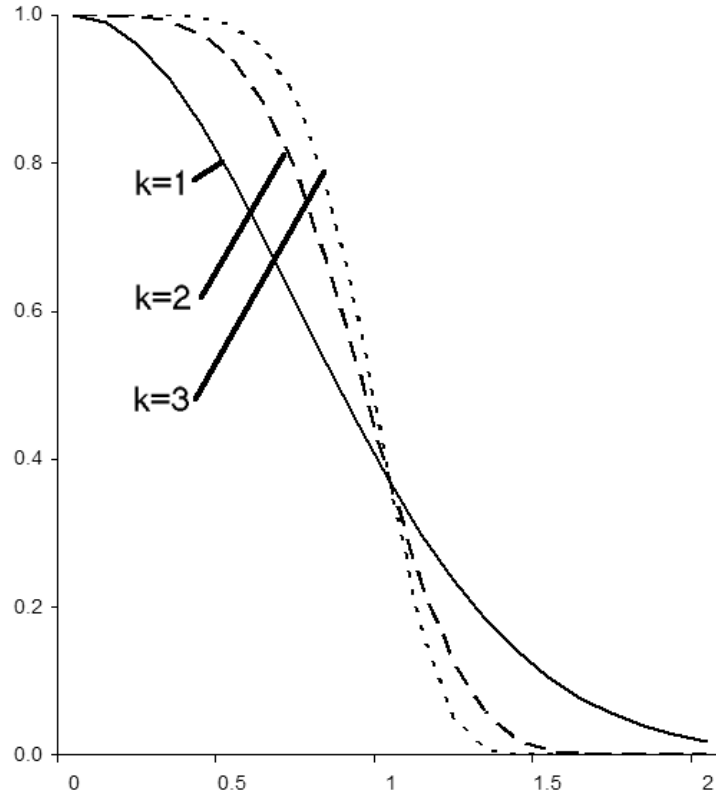


Figure 5.5: Degradation of Trust Value of 1 at different rates.

decay rate of the trust value. As k increases, the trust value decays slowly at first and then rapidly decays, but we see that for $k > 1$ the value approaches 0 between about $t_{1.25}$ and $t_{1.5}$. Even with $k = 1$ we see that by t_2 the decayed value is approaching 0. With higher values of k the decay approaches zero between t_1 and $t_{1.25}$.

Time periods are arbitrary, as the time difference between t_0 and t_1 could be 10 seconds or 2 months. The *period of decay* (the time from t_0 and t_1) should be set based on how often trust values are calculated, so that most calculations of the value of a trust relationship happen before the value of a previous trust vector has decayed to nearly zero. This is not to say that trust should not be calculated using a previous

trust vector that has decayed to nearly zero. An example is an agent that had not reported data for a long period of time (“long period of time” being subjective to the system), and the agent is now sending data again. If the agent was highly trusted before going silent for the long period, the new data should not be trusted at the same level as the agent may have been compromised (and fed misleading data) or may not have the same capability as it once had (e.g., the agent was reporting from inside a firewall, but the host computer was moved outside the firewall, and the agent does not have the same access to information that it once had). Thus the new data should be treated as it came from an agent that is untrusted (trust value of 0).

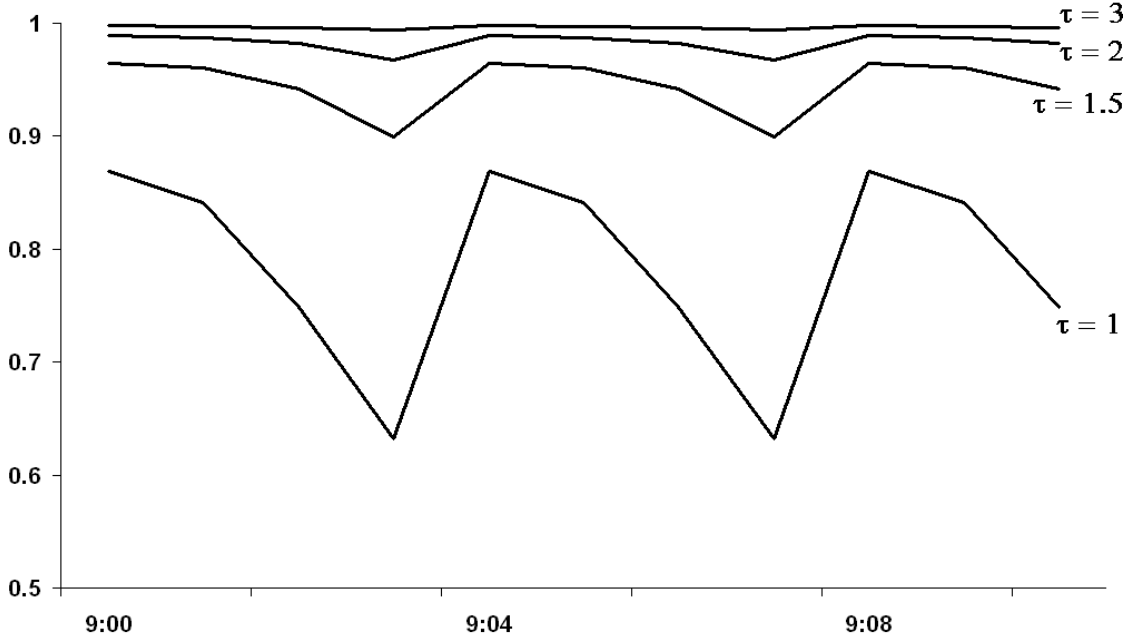


Figure 5.6: Degradation of trust at different levels of the *period of decay* (τ).

5.2.2 Results. Figure 5.6 shows the effect of the different levels of the *period of decay* (τ) at steady state. All the agents interpret the data the same, so

all agents are generating current trust levels of 1, representing complete trust with each other. Because the Time-dependant weight was $0.\bar{3}$, one third of the value of the trust relationship is based on the degraded time-dependant trust. Figure 5.6 shows that when the period of decay is too close to the exchange rate, the trust degrades too fast to be used effectively. Using Figure 5.4 as a reference, when $\tau = 1$ times the exchange rate, the time dependant value has degraded to the value at t_1 .

VI. Conclusions and Future Work

This thesis presented the hypothesis that the Trust Vector model is an acceptable model for the distributed environment of the CyberCraft fleet. This research supports the hypothesis that the Trust Vector model can be modified to fit the CyberCraft Initiative. This research proposed some modifications and expansions to the Trust Model Vector (listed in Chapter III), and identified areas for future research (Section 6.2)

Contributions: This research identifies that the *transactional paradigm* models the environment of the CyberCraft fleet better than a *synchronous paradigm*. A modification to the degradation function is proposed (Section 5.2.1), as well as a modification to the evaluation of the events (defining the benefit of each event as a value between $[-1, 1]$, rather than just as a trust-negative or trust-positive event (Figure 3.6)). Experiment I (Section 6.1.1) identifies the degrading value of older intervals in the experience component. Experiment II (Section 6.1.2) identifies problems with the degradation function, both with the function itself and the need for defining a reasonable *period of decay*.

Shortcomings: There are a few remaining shortcomings of the Trust Vector model that still need to be addressed. The largest of these shortcomings is the amount of data needed to populate a trust vector. In the transactional paradigm, an agent needs several transactions to build enough history to make accurate evaluations about the trustworthiness of a remote agent. Because the Trust Vector model does not rely

on experience alone, the model is able to rely on the other two components to build a more robust picture of trust than by relying on experience alone.

Another area of research for the implementation of the Trust Vector model is how to evaluate an event if no corroborating data is present. The use of Hysteresis (Section 3.1.2.2) to force other agents to corroborate is useful only if other agents have the same access to the environment that the reporting agent had. In some situations, only one agent has access to the segment of the environment it is reporting on. Markov Decision Processes (see Section 6.2.11) and other prediction mechanisms may be a way to combat this.

The Trust Vector model did not address how the knowledge component should be set, which may be a strength in allowing the implementor to determine how best to use the knowledge component. Future work on how best to utilize the knowledge component is beneficial to the CyberCraft Initiative and other implementors of the Trust Vector model.

6.1 Experiments

The conclusions drawn from the results of the Experiments are as follows:

6.1.1 Experiment I: Limits of Historical Data for Calculating the Experience Component. **Conclusion:** As the impact of the oldest interval decreases, even with a 50% weight of the experience component on the trust value, keeping 9 intervals instead of 10 intervals leads to less than a 2% maximum difference, which is below

the level of granularity needed for most applications. When CyberCraft is deployed across thousands of computers and each agent holds multiple trust vectors per remote agent, keeping one less byte of data per trust vector may be significant.

6.1.2 Experiment II: Utility of Degradation Function. **Conclusion:** The utility of previous Trust Values decreases as time increases, and Section 5.2.1 shows that by t_2 (two time periods since the trust value was last calculated) previous trust values have decayed to the point that they are too small to impact the current value of trust. This conclusion is valid using the modified equation $\mathbf{v}(T_{t_n}) = \mathbf{v}(T_{t_i})e^{-(\mathbf{v}(T_{t_i})^{-1}\Delta t)^{2k}}$, as the original equation keeps the value of smaller initial trust longer (an initial trust value of 0.25 decays to 0.05 at approximately $t_{4.5}$ at $k = 2$).

The period of decay must be no less than 1.5 times as long as the expected time between trust calculations. In the *transactional paradigm*, if data posted by an agent is expected to be read every 8 minutes, then the period of decay must be no less than 12 minutes.

6.2 Future Work

This section proposes areas for future research in fitting the Trust Vector model to the CyberCraft Initiative, as well as other areas that the Trust Vector model may be applicable to.

6.2.1 Transactional Paradigm. This research uses the *synchronous paradigm* for both experiments, and is not able to explore the *transactional paradigm* outlined

in Section 3.1.2. Future research must explore the use of the Trust Vector model in the *transactional paradigm* to identify other challenges with the use of the modified Trust Vector model.

6.2.2 Number of Agents in a Recommendation Pool. This research is focusing on the application of the Trust Vector model to the CyberCraft Initiative, to include the balancing the number of agents that check each other with the overhead of storage and network traffic that larger trust groups cause. As the CyberCraft Initiative will eventually span hundreds of thousands of computers, scalability of the Trust Vector model becomes an issue.

6.2.3 Abstraction of Trust Context. Another scalability issue that I was unable to address in this research is the abstraction of the context of trust relationships, where the trade off between the granularity of multiple trust vectors for specific tasks must be balanced with the storage and computational complexity requirements to support several trust vectors.

An example of this is combining the trust relationships for several specific contexts into a generic context. It provides more granularity for a network scanner to have different trust relationships for each vulnerability scanned, as some scanners have better detection rates against some vulnerabilities than other vulnerabilities. The downside of having a trust vector for each vulnerability is that with a large number of vulnerabilities, the storage for each trust vector becomes significant. To balance this, the system may abstract the vulnerabilities into types of vulnerabili-

ties, such as grouping vulnerabilities by network layer or by year discovered. Thus a network scanner may have one trust vector for each network layer rather than the thousands of trust vectors for each vulnerability.

Future research must address the culminating point between the granularity of trust contexts and the storage, computational power, and bandwidth needed to support multiple trust vectors for different contexts between agents.

6.2.4 Experience Component. The effects and challenges of a modification of the experience component's intervals to be fixed time periods instead of fixed width based on number of events is not addressed by this research. One challenge is how to treat intervals without events. Should intervals without events be treated as trust-neutral (0) for the calculation of the experience component, or should they not be factored in, and the intervals with events are normalized in a similar fashion to recommendations? This is an area for future work.

6.2.5 Knowledge Component. This research did not address the settings of the knowledge component, and arbitrarily set the value at 1. Future research for the knowledge component includes how to select an initial setting for the component that is meaningful. One method for doing this was discussed in Section 3.1.1, using a formal validation of a component to give it an initial value.

Future work may also include how to dynamically change the value of the knowledge component. Chakraborty and Ray's work on privacy [7], where they split the knowledge subcomponents apart may be a good starting point.

6.2.6 Normalization of Recommendations.

It was identified in Section 2.2.2.3 that the equation:

$$\Psi R_{Bob}^c = \frac{\sum_{j=1}^n |\mathbf{v}(Alice \xrightarrow{rec} j)_t^N| \cdot V_j}{\sum_{j=1}^n |\mathbf{v}(Alice \xrightarrow{rec} j)_t^N|}.$$

had problems in accomodating negative trust values for recommendations. If *Alice* distrusts *Bob*'s recommendation, this equation calculates his recommendation as if *Alice* trusted it.

Table 6.1 show modifications to the normalization equations, and Table 6.2 shows the effects of these modifications to the equation when normalizing recommendations. ΨR_{Bob}^c represents *Alice*'s recommendation component for *Bob*, where *Alice* receives recommendations from *Cathy* and *David*. *Alice* trusts *Cathy*'s recommendations at 0.5, and trusts *David* at -0.5 (indicating distrust).

Table 6.1: Modifications of the normalization equation.

Equation Number	Equation $\Psi R_{Bob}^c =$
Original	$\frac{\sum_{j=1}^n \mathbf{v}(Alice \xrightarrow{rec} j)_t^N \cdot V_j}{\sum_{j=1}^n \mathbf{v}(Alice \xrightarrow{rec} j)_t^N }$
Modification 1	$\frac{\sum_{j=1}^n \mathbf{v}(Alice \xrightarrow{rec} j)_t^N \cdot V_j}{\sum_{j=1}^n \mathbf{v}(Alice \xrightarrow{rec} j)_t^N }$
Modification 2	$\frac{ \sum_{j=1}^n \mathbf{v}(Alice \xrightarrow{rec} j)_t^N \cdot V_j }{\sum_{j=1}^n \mathbf{v}(Alice \xrightarrow{rec} j)_t^N }$
Modification from p-Trust [7]	$\frac{\sum_{j=1}^n \mathbf{v}(Alice \xrightarrow{rec} j)_t^N \cdot V_j}{\sum_{j=1}^n \mathbf{v}(Alice \xrightarrow{rec} j)_t^N}$

It appears that Modification 1 $(\frac{\sum_{j=1}^n \mathbf{v}(Alice \xrightarrow{rec} j)_t^N \cdot V_j}{\sum_{j=1}^n |\mathbf{v}(Alice \xrightarrow{rec} j)_t^N|})$ makes the most sense, as receiving positive recommendations from a trusted agent and from a distrusted agent cancel each other out, and receiving a negative recommendation from a distrusted agent is interpreted as a positive recommendation from a trusted agent. Use of the

Table 6.2: Effect of different normalization equations.

Equation $\Psi R_{Bob}^c =$	<i>Cathy's</i> Recommendation $\mathbf{v}(Alice^{rec}Cathy)_t^N = 0.5$	<i>David's</i> Recommendation $\mathbf{v}(Alice^{rec}Cathy)_t^N = -0.5$	Results
Original	0.75	0.75	0.75
Original	-0.75	0.75	0.0
Modification 1	0.75	0.75	0.0
Modification 1	-0.75	0.75	0.75
Modification 2	0.75	0.75	0.75
Modification 2	-0.75	0.75	0.75
p-Trust Modification	0.75	0.75	Undefined (Divide by 0)
p-Trust Modification	-0.75	0.75	Undefined (Divide by 0)

p-Trust equation with both positive and negative trust values can lead to an answer that is greater than 1 or is undefined.

This research did not have time to explore the use of these modifications, but exploring this area is useful as future research.

6.2.7 Additional Components. Ray and Chakraborty [17] state that the three components of their Trust Vector model are not the only ones that may be used, and this model can be expanded again to add a fourth or fifth component, or an existing component can be replaced. Chakraborty and Ray's expansion of the Trust Vector model into the p-Trust model [7] uses the two subcomponents of the knowledge component as components of the p-Trust vector. Future work in this area may prove beneficial to the Trust Vector model as well as other trust models.

6.2.8 Differing Views of Data. Another area of future research is the use of Trust Vectors with differing views that do not necessarily conflict. An example of this is in the OS Fingerprinting scenario in Section 4.3, if one agent reports a target system as being a Mac OS, another agent reports the same system as a Windows OS (but not specifically XP or 2000), and a third machine reports the machine as a Windows XP SP2, how does the model combine these inputs to produce a best guess at the target machine's OS with an associated confidence. Further research entails how to accommodate data that changes with time, such as a dual booting system or virtual machines.

6.2.9 Ambiguity of Data. Ambiguity in the reported data is another area of future research. An example of this is if an agent was fingerprinting a target system and the data received from the scan was not conclusive, e.g., the target OS had some Linux characteristics, but not enough to definitely label the target system as Linux. Another example of the above is if the agent suspects that it is compromised, and is being fed inaccurate data. Data sent from these agents to other agents must reflect the ambiguity in the data held by the reporting agent. This ambiguity can be reflected in a confidence tag included with the data.

6.2.10 Integration of VTrust into the CyberCraft Initiative. This research did not address the user of the VTrust system as the Trust Management framework for the CyberCraft Initiative. This research instead presumed that trust data is stored at each agent, and that trust is shared by querying other agents.

Future research must address the tradeoffs and vulnerabilities of storing all trust data in a central database. Using VTrust, agents do not need to store trust data locally, therefore the challenge of the scalability of data storage is transferred to the database. The trade-off is that all agents need to query the database for trust calculations.

Another area of research in central storage of trust data is the amalgamation of trust data. In the distributed model for trust data, the experience component for trust relationships was based on only the local agent's experience with a remote agent. By using a central repository for all trust data, the experience components could be calculated by all agents' recent experiences with the remote agent.

6.2.11 Lack of Corroborating Data and Markovian Decision Processes. Another area of research for the implementation of the Trust Vector model is how to evaluate an event if no corroborating data is present. The use of Hysteresis (Section 3.1.2.2) to force other agents to corroborate is useful only if other agents have the same access to the environment that the reporting agent had. In some situations, only one agent has access to the segment of the environment it is reporting on.

One possibility of dealing with integrating Markovian Decision Processes [6] or another predictive mechanism into the Trust Vector model to supplement building a trust vector with little data.

6.2.12 Dynamic Routing and Topology Control. Another possible application of Trust Vectors is the use of Trust Vectors with dynamic routing of network

traffic. This may include using the rate of dropped packets for the experience component, the available bandwidth of a link (as well as graph theory with network topology) as the knowledge component, and abilities of neighboring routers to reach a distant end for the recommendation component to determine the utility of a link to reach a distant end. The Air Force Institute of Technology (AFIT) has been researching this area in depth, and may benefit from collaborating with Colorado State University in researching the applicability of the Trust Vector model to dynamic routing.

6.2.13 Evolutionary Algorithms and Artificial Immune Systems. Capt Charles “Space” Haag’s thesis [9] explores the use of evolutionary algorithms to build better artificial immune systems (AIS). The “anti-bodies” of the AIS are trained to recognize the strings inherent in the normal state of the system, and then are introduced to non-state strings, which may be malicious attacks that the AIS is trying to block. The anti-bodies that perform well in determining state vs. non-state are heuristically selected to survive and spawn more anti-bodies, while the anti-bodies that perform poorly are removed. While not a distributed system, it is possible that the Trust Vector model could be integrated into the anti-bodies to give the system a better sense of which anti-bodies are performing well.

6.2.14 The “High Ground” of Cyberspace. While this area of research does not directly relate to the CyberCraft Initiative or the use of Trust Vectors, identifying the “high ground” of Cyberspace is pertinent to how the USAF and U.S. military wages Cyber-warfare in the near future. Alvin and Heidi Toffler identified the concept

of “high ground” in space as the lunar Lagrangian points L4 and L5 [20]. These points are located where the gravity between the earth and the moon are equal, thus a satellite or space station expends little energy to remain parked at one of these points, while an opponent has to expend a great amount of energy to reach these points to attack the satellite located there. This concept is analogous to the high ground of a terrestrial battlefield, where a defensive position on top of a hill enables the defender to rain arrows or bullets upon an opposing force assaulting the defender’s position, while the opposing force is expending energy climbing the hill.

Identifying the “high ground” for the Cyber-battlefield now allows up to posture our Cyber-forces on the “high ground” before we are engaged in battle. I hypothesize that trust is a key component of the “high ground” of Cyberspace, as it forces the opponent to expend energy to overcome trusted relationships to assault the defender.

Bibliography

1. “eBay® - Feedback Forum”. URL <http://pages.ebay.com/services/forum/feedback.html>.
2. “Open Systems Joint Task Force - Terms and Definitions”. www.acq.osd.mil/osjtf/termsdef.html.
3. Abdul-Rahman, Alfarez and Stephen Hailes. “A Distributed Trust Model”. *Proceedings of the 1997 workshop on New Security Paradigms*, 48–60),. 1997.
4. Abdul-Rahman, Alfarez and Stephen Hailes. “Supporting Trust in Virtual Communities”. *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences (HICSS-33)*, Maui, Hawaii, USA, 1769–1777),. 2000.
5. Beth, Thomas, Malte Borchering, and Birgit Klein. “Valuation of Trust in Open Networks”. *Proceedings of the 3rd European Symposium on Research in Computer Security (ESORICS’94)*, Brighton, U.K., 3–18. 2004.
6. Boutilier, Craig, Thomas Dean, and Steve Hanks. “Decision-Theoretic Planning: Structural Assumptions and Computational Leverage”. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.
7. Chakraborty, Sudip and Indrajit Ray. “Allowing Finer Control Over Privacy Using Trust as a Benchmark”. *7th Annual IEEE Information Assurance Workshop (IAW’06)*, United States Military Academy, West Point, NY, USA. 2006.
8. Dubrawsky, Ido. “Effects of Worms on Internet Routing Stability”, June 2003. URL <http://www.securityfocus.com/infocus/1702>.
9. Haag, Charles. *Evolutionary Algorithms and Artificial Immune systems*. Master’s thesis, Air Force Institute of Technology, 2007.
10. Jabbour, Kamal. “Cyber RMA”, 2007.
11. Jabbour, Kamal and David Bibighaus. “CyberCraft Information Workshop”. June 2006.
12. Jøsang, Audun. “The right type of trust fo distributed systems”. *Proceedings of the 1996 workshop on New Security Paradigms*. 1996.
13. NIST/SEMATECH. “NIST/SEMATECH e-Handbook of Statistical Methods”, February 2007. URL <http://www.itl.nist.gov/div898/handbook>.
14. Phister, Paul W., Jr., Dan Fayette, and Emily Krzysaik. “CyberCraft: Concept Linking Network Control Warfare Principles with the Cyber Domain in an Urban Opertional Environment”. Air Force Research Labratory, Information Directorate, 2004.

15. Purser, Steve. "A Simple Graphical Tool For Modelling Trust". *Computers & Security*, 20:479–484, 2001.
16. Ray, Indrajit and Sudip Chakraborty. "Cyber Trust Research: Motivation (in a Nutshell)". URL <http://www.cs.colostate.edu/~indrajit/Security/trust/index.htm>.
17. Ray, Indrajit and Sudip Chakraborty. "A Vector Model of Trust for Developing Trustworthy Systems". *Proceedings of 9th European Symposium on Research in Computer Security (ESORICS'04), Sophia Antipolis, France*. 2004.
18. Ray, Indrajit, Sudip Chakraborty, and Indrakshi Ray. "VTrust: A Trust Management System Based on a Vector Model of Trust". *Proceedings of 1st International Conference on Information Systems Security (ICISS'05), Sophia Antipolis, France*. 2005.
19. Rowstron, Antony and Peter Druschel. "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems". *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany*, 329–350. 2001.
20. Toffler, Alvin and Heidi. *War and Anti-War*. Warner Books, 1993.
21. Trusted Computing Group. "Embedded Systems and Trusted Computing Security". https://www.trustedcomputinggroup.org/groups/tpm/embedded_bkgdr_final_sept_14_2005.pdf, 2005.
22. Wynne, Michael and T. Michael Mosely. "Letter to Airmen: Mission Statement". <http://www.af.mil/library/viewpoints/jvp.asp?id=192>, December 2005.
23. Xiong, Li and Ling Liu. "A Reputation-Based Trust Model for Peer-to-Peer eCommerce Communities". *Proceedings of IEEE Conference on E-Commerce (CEC'03)*, 275–284. 2003.
24. Yahalom, R., B. Klein, and T. Beth. "Trust Relationships in Secure Systems: A Distributed Authentication Perspective." *Proceedings of the IEEE Computer Society Symposium on Security and Privacy, Oakland, California, USA, IEEE Computer Society*, 150–164. 1993.

REPORT DOCUMENTATION PAGE					<i>Form Approved OMB No. 0704-0188</i>	
<small>The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.</small>						
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.						
1. REPORT DATE (DD-MM-YYYY)		2. REPORT TYPE			3. DATES COVERED (From - To)	
4. TITLE AND SUBTITLE				5a. CONTRACT NUMBER		
				5b. GRANT NUMBER		
				5c. PROGRAM ELEMENT NUMBER		
6. AUTHOR(S)				5d. PROJECT NUMBER		
				5e. TASK NUMBER		
				5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)					8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)					10. SPONSOR/MONITOR'S ACRONYM(S)	
					11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT						
13. SUPPLEMENTARY NOTES						
14. ABSTRACT						
15. SUBJECT TERMS						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON	
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER (Include area code)	